Deliberative Agent Model of a Taxi Rank

by

Jurie Heinrich Venter

Thesis presented in fulfilment of the requirements for the degree of

Master of Engineering (Civil)

in the Faculty of Engineering at Stellenbosch University

Supervisors: Dr. S. J. Andersen
Dr. G. C. van Rooyen

December 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Abstract

Deliberative Agent Model of a Taxi Rank

Jurie Heinrich Venter

Department of Civil Engineering
University of Stellenbosch
Private Bag X1, Matieland 7602 , South Africa.

Thesis: MEng (Civ)

December 2019

Minibus-taxi loading facilities known "taxi ranks" in South Africa are designed according to guidelines prescribed by the South African Department of Transport. Currently, it is only possible to observe whether designs are successful once they have been constructed. This research project aims to develop a tool that can provide answers to "what-if?" questions posed about taxi rank designs so that the risks associated with unsuccessful designs can be reduced. Taxi rank designers can then use the tool in addition to the guidelines. It is proposed that the development of a detailed simulation model will allow taxi rank designers to test the effects of their ideas without the risks associated with building a taxi rank. A simulation model incorporating deliberative agents and three-dimensional geometry was developed. The model, referred to as ACTS, was used to create a digital twin of the Bergzicht taxi rank located in Stellenbosch. ACTS produces results within an acceptable margin of error and it is proposed that it is used as part of the taxi rank design process.

Uittreksel

Deliberative Agent Model of a Taxi Rank

Jurie Heinrich Venter

Department of Civil Engineering
University of Stellenbosch
Private Bag X1, Matieland 7602 , South Africa.

Tesis: MEng (Civ)

Desember 2019

In Suid-Afrika, word sogenoemde 'taxi ranks' ontwerp volgens riglyne wat deur die Suid-Afrikaanse Departement van Vervoer voorgeskryf is. Tans is dit slegs moontlik om te sien of ontwerpe suksesvol is sodra dit gebou is. Hierdie navorsingsprojek het ten doel om 'n hulpmiddel te ontwikkel wat antwoorde kan gee op 'wat-as?' vrae rakende ontwerpe vir taxi ranks. So 'n hulpmiddel kan die risiko's verbonde aan onsuksesvolle ontwerpe verminder. Taxi rank ontwerpers kan dan die hulpmiddel benewens die riglyne gebruik. Daar word voorgestel dat die ontwikkeling van 'n gedetailleerde simulasiemodel taxi-rank ontwerpers in staat sal stel om die gevolge van hul idees te toets sonder die risiko's verbonde aan die oprig van 'n taxi rank. 'n Simulasiemodel wat beraadslagende sagteware agente en driedimensionele geometriese modelle bevat, is ontwikkel. Die model, wat ACTS genoem word, is gebruik om 'n digitale tweeling van die Bergzicht taxi rank in Stellenbosch te skep. ACTS lewer resultate binne 'n aanvaarbare foutmarge en daar word voorgestel dat dit gebruik word as deel van die ontwerp proses van taxi ranks.

Acknowledgements

I would like to express my sincere gratitude to my supervisors Dr. S. J, Andersen and Dr. G. C. van Rooyen. Their guidance and motivation were invaluable during the process of identifying and completing this thesis.
I would also like to thank Stellenbosch municipality for their cooperation in obtaining the data needed to test the model developed.
Finally, I would like to thank all the mentors and teachers that have shaped and brought me here.

Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

ACTS Agent Centred Taxi-Transit Simulation
CAD Computer Aided Design
CSV Comma Separated Values
FSM Finite State Machine
GOAP Goal Oriented Action Planning
OOP Object Oriented Programming
STRIPS Stanford Research Institute Problem Solver
SVG Scalable Vector Graphics
UML Unified Modelling Language
XML eXtensible Markup Language

Chapter 1

Introduction

1.1 Research area

The research described here relates to the informal public transport space in South Africa. Over the course of the last 50 years, the minibus taxi industry has taken a leading role in this space. It provides services to thousands of commuters in South Africa. The results of the General Household Survey of 2017 indicate that 22.9% of work bound trips and 6.6% of school bound trips are made by taxi. In addition, they showed that 37.1% of households had one or more members that used a taxi in the week before the survey was conducted (Statistics South Africa 2017). The scale of the industry and the valuable contribution it makes to meeting the transportation needs of South African citizens shows the importance of finding ways to improve it. This research will focus on design and layout of the loading areas for minibus taxis, colloquially referred to as "taxi ranks". An agent-based simulation model will be built that models the behaviour of pedestrians, commuters and the mini-bus taxis themselves. It is hoped that such a model will aid taxi rank designers who want to answer "what-if?" questions. Such question may include: What would happen if the rank geometry is changed? What would happen if the rules of operation at the rank were changed?

1.2 Motivation

In its official guidelines for the design of minibus taxi loading (South African Department of Transport 2007), the Department of Transport identifies a deficit in the amount of research in this field. The guidelines provide only two options with regards to layout: parallel island and oval island layouts. In addition, the demand and loading patterns, specific to the rank to be designed, are not considered (van-Biljon and CJ Venter 2013). Van Biljon and Venter developed a stochastic simulation model that allowed them to test the effects of passenger volumes, fleet size and the queuing behaviour of passengers on the

effectiveness of a minibus taxi loading facility. In their recommendations, they strongly recommended that the behaviour of the agents in the environment of a facility be incorporated in further research. This project will aim to expand upon the work done by van Biljon and Venter by creating an agent-based simulation model for minibus taxi loading facilities. An agent-based simulation will allow investigations into the experiences of the commuters in a hypothetical design as well as explore factors that affect the effectiveness of the facility beyond those that can be tested in a stochastic model.

1.3 Problem statement

The minibus taxi loading area or "taxi rank" is an integral part of the taxi system in South Africa. This system transports thousands of commuters daily. The design of such loading areas has a direct effect on the experiences of commuters and the effectiveness of the system. The guidelines provided by the National Department of Transport are currently based on a simplified deterministic method that makes the following assumptions:

- There is one taxi using a taxi berth, one inbound and one outbound, giving a ratio of 1 berth for every 3 taxis in the fleet. This is problematic as it assumes that the number of taxis in the fleet is known.
- The lengths of the routes taken by taxis are about the same, which is not the case in reality
- The taxis arrive and depart in uniform intervals, which is also not the case in reality

To improve upon the design process, less reliance needs to be placed on assumptions and the recommendations made by the guidelines need to be tested. It is inefficient and expensive to keep building facilities and hoping to learn from each experience and the functioning of the built facility. It is postulated that the development of a computer application that uses an agent-based simulation model will help address these shortcomings. With such an application, designers can test the effects of various configurations on the effectiveness of a facility and improve the experiences of the users before the facility is built.

1.4 Research Objectives

The following objectives were identified:

- Report on the broad context of the informal public transport sector in South Africa

- Report on the role that the minibus taxi loading facility plays in this context
- Report on the current methods used to design minibus taxi loading facilities
- Report on agent-based simulation
- Identify the agents that play a role in a minibus taxi loading facility
- Research and model their behaviour and decision-making framework
- Research and model the environment these agents will function in
- Create a simulation model incorporating agents present at taxi ranks
- Compare the created simulation model to existing observations at taxi ranks
- Determine the validity of the simulation model
- Report on the process followed to create the simulation model and application

An investigation into existing literature was made with these research objectives in mind. It can be found in the following chapter.

Chapter 2

Literature Review

This section aims to synthesise the current state of research in the study area identified in chapter 1. It is divided into three parts; an introduction, a discussion section and a concluding section. The topics to be explored are the informal public transport sector in South Africa, the minibus taxi loading facility and simulation with an emphasis on studies and techniques that relate to minibus taxi loading facilities.

2.1 Informal Transport

The term informal transport refers to transport services provided by entities that operate outside of the control of the government. Informal transport may take the form of service providers that use minibuses, sedans as taxis or motorcycles to transport members of the public (Cervero and Golub 2007). Such service providers usually start operating in low-income, dense, urban areas where state sanctioned formal service providers such as bus services cannot meet the demand generated in those areas. The informal service providers identify and fill the gaps that formal service providers cannot. Because informal transport industries arise spontaneously and without oversight from regulating authorities, they initially tend to display the following characteristics:

- Operation without appropriate licenses
- Operation with potentially unroadworthy vehicles
- Operation with unqualified drivers
- The use of violence to assert control over areas
- Anti-competitive practices such as:
  - Functioning as cartels

  - Keeping markets captive
  - Price fixing

Developing countries provide the perfect conditions for informal transport sectors to arise. (Cervero and Golub 2007) identify four examples of informal transport industries in developing countries:

- Political strife in the 1970s caused the bus system in Mexico City to operate with reduced and inadequate capacity. The authorities then opened the industry to informal operators to meet the demand that the formal bus system could not meet anymore. By the 1980s, the formal bus system in Mexico City had ceased to function and the number of informal operators surged as a result. By the 90s, trips by minibuses affiliated with the informal transport sector accounted for 50% of trips made in the city. It was estimated that only about half of the service providers were licensed by the late 90s (ibid.). Public transport initiatives by the authorities has since resulted in a resurgence in the formal transport sector in Mexico City (Cocking 2017).

- Many different types of vehicles including minibuses, vans and pickup trucks are used by informal transport service providers in Kenya's capital city, Nairobi. They serve a third of the public transport demand there. Because their contribution to the city's transport needs is so substantial, they were legalised in 1984. Their operations however, remain unregulated (Cervero and Golub 2007). (Vural 2019) confirms that Cervero & Golub's assessment of Nairobi's public transport situation is still valid.

- To travel around in Jakarta, Indonesia, commuters can choose to use cycle rickshaws, tuk-tuks or minibuses. The smaller transport modes can usually be found in areas of the city with narrow streets or on the edges of the city. This arrangement was encouraged by the city authorities to promote safety and provide a more orderly environment for the bigger modes such as minibuses to operate. The city authorities regulate the bigger transport modes to ensure that the vehicles used are roadworthy, fares are fair, and schedules are in place. (Cervero and Golub 2007)

- The minibus taxi industry in South Africa whose history and operations are of interest for this research project and explained in the next sections.

2.2 South African minibus taxi industry

The minibus taxi industry fills the role of a public transport provider in South Africa. The industry operates as a conglomeration of privately owned informal businesses and is not subsidised by the South African government (van-Dalsen

2018). It has its roots in the late 1970s when there was a high demand to transport the workforce on the outskirts of urban areas to their places of work in the centres of urban areas. To help meet this demand, The Road Transportation Act of 1977 deregulated the taxi industry and allowed the use of minibuses as taxis that could be waved down (Ingle 2009). New regulations also increased the number of passengers that a driver could transport from 9 to 15. The adherence to operation in prescribed areas was also decreasingly monitored and enforced.

Entrepreneurship flourished during this period of low supervision and the industry grew very quickly. The system that emerged was one where taxi owners would appoint taxi drivers without formal contracts. This opened the door to abusive labour practices and drivers typically worked unreasonably long hours for the remuneration that they received (ibid.) (Fobosi 2013). Violence between competing taxi operators also became a major problem in this time.

After the elections of 1994, the government started to focus on the taxi industry and efforts have been made to formalise and regulate the industry. The goal of this focus is to improve the safety of the taxi mode, its affordability and to ensure that acceptable contracts between employers and employees are in place (ibid.). In 2005, the government also started to enact its taxi recapitalisation programme in which it planned to spend 7.7 billion rand over 5 to 7 years to replace the ageing, unsafe vehicles in the taxi fleet with vehicles that adhere to stricter safety standards. The labour conditions of drivers also received attention (Ingle 2009). As of 2018, the programme is still on track and the government allocated 389 million rand toward the programme in the 2017/2018 budget according to (van-Dalsen 2018). (Woolf and J W Joubert 2013) however, state that the taxi recapitalisation programme has been considered a failure as the government did not get adequate engagement from the taxi industry in the programme. (Christoffel Venter 2013) cites the reason for this lacklustre engagement as the misguided efforts of the government to interact with the industry by using a "top-down" approach. This top-down approach manifested itself as an insistence that the industry represent itself as one union.

An alternative to the approach of treating the taxi industry as a loosely bound whole is proposed by (Woolf and J W Joubert 2013). They argue that the taxi industry should be viewed through the lens and perspective of the individuals that make it up. They call this a "People-centred view". Guided by this principle, an understanding for the needs of individual role-players in the industry can be developed. Targeted "bottom-up" initiatives can then make real change for the industry possible.

2.3 Minibus taxi loading facilities

Minibus taxi loading facilities, colloquially known as "taxi ranks" are the hubs around which the taxi industry structures itself. At taxi ranks, there are role

players present including: the commuters, the taxi drivers, the queue marshals and hawkers. The rank generally begins to operate before 5:00 am, when the taxis arrive and are allocated berths by the queue marshal. The queue marshals know the taxi drivers and the routes that they will take well. They show commuters where to queue for the various routes that depart from the rank. If there is more than one taxi that services a route, the queue marshal will usually direct the commuters to fill the taxi that arrived first. In that way, they are also responsible for the queuing of the taxis in the loading facility. The peak periods at taxi ranks are generally between 05:00 and 10:00 am and again between 14:00 and 19:00. The last taxi usually leaves the rank at about 19:00. Hawkers are present throughout the day. (Masuku 2016). At taxi ranks where there are no queue marshals present, the routes that the taxis will take is spread by word of mouth or signage. The physical geometry and structure of taxi ranks are usually designed according to the guidelines provided by the National Department of Transport. The important general design principles contained in the guidelines are as follows:

- Passengers should board and alight on the left hand side of the vehicle.
- One way movements are preferred as they are safer and more easy to control than two way movements.
- There should be no need for a vehicle to reverse in the facility. The need to reverse at parking bays is the only exception to this rule.
- The areas where passengers are to queue should be clearly delineated. Walkways should preferably be paved, raised and kerbed areas. The walkways must be clearly separated from the areas meant for vehicles.
- Passengers that have mobility impairments must be accommodated. Kerb ramps are an example where this is applied.
- Throughout the facility, vehicles and pedestrians must be provided with adequate sight distance.
- The locations where pedestrians and vehicles enter or exit the facility must be placed such that the conflicts between pedestrians and vehicles are minimised. Where conflicts cannot be avoided, they should be arranged such that vehicles are likely to be driving slowly.

These guidelines base the number of berths to be provided on a deterministic method that makes the following assumptions:

- The lengths of the routes taken by taxis are about the same
- The taxis arrive and depart in uniform intervals

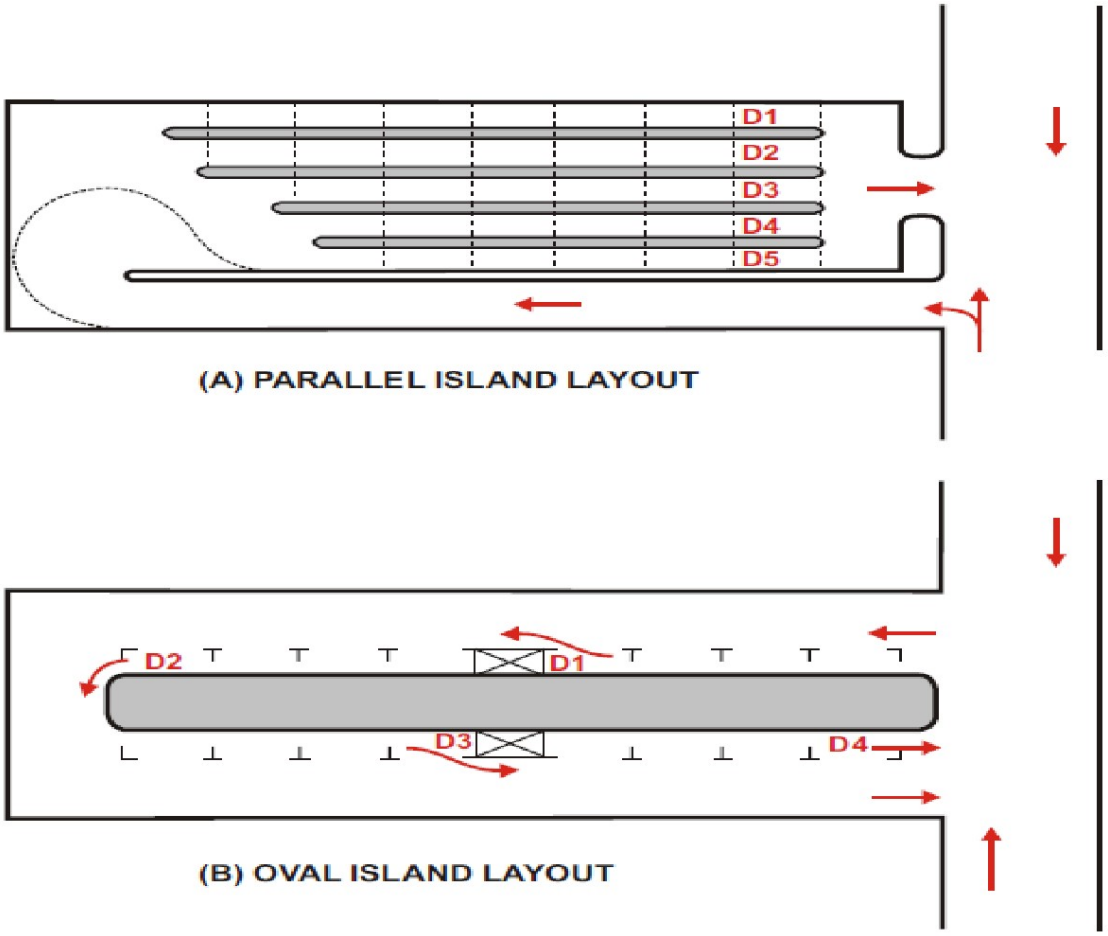


Figure 2.1: Different proposed NDoT layouts for minibus taxi loading facilities

- There is one taxi using a taxi berth, one inbound and one outbound, giving a ratio of 1 berth for every 3 taxis in the fleet.

Two different layouts are proposed in the guidelines, namely; the parallel island layout and the oval island layout. These can be seen in figure 2.1. The number of lanes provided is usually determined by the number of destinations that are to be served by the taxi rank (South African Department of Transport 2007). (van-Biljon and CJ Venter 2013) find the technique specified in the guidelines inadequate. Their reasoning is that the current approach requires the number of taxis in the fleet for an area to be known. For a taxi rank that has not been built yet as well as existing taxi ranks, this number is not known exactly. Furthermore, the current design methodology assumes routes of similar length and uniform arrival and departure times. These assumptions do not hold up. Additionally, the impact of the geometry and layout of the facility is not taken into consideration. To address these issues, they suggest

the use of simulation. Historical applications of simulation techniques in the taxi industry environment is discussed in the next section.

2.4 Traffic simulation

A traffic simulation is a model of what is expected to occur in reality for a given traffic problem. Usually, simulations are mathematical or logical abstractions of the real world that have been implemented as computer programs. Traffic problems however, comprise complex processes in which many different components interact to produce a general outcome. These complex processes do not lend themselves to acceptable descriptions in mathematical or logical terms. The focus of traffic simulation models is therefore placed on the individual components of the problem as these can be described to an acceptable degree (Mathew 2017). These components are then allowed to interact, and the complex processes desired then arise naturally. (Lieberman and Rathi 2005) provide a list of applications for traffic simulation models:

- To compare the effects of proposed treatments and their alternatives. An example of this is testing various traffic signal strategies.
- To test new designs. Before constructing a new transport facility such as a road, bus stop or of interest to this research project: a taxi rank, various designs can be simulated and compared. This ensures that money is not wasted in constructing a design that will prove sub-optimal.
- To aid the design process. By looking at the graphical and numerical output of a simulation model, a designer can identify flaws or areas that can be improved in their design.
- To visualise the output of other tools.
- To train personnel. Simulation models offer the opportunity for personnel to be exposed to the conditions they will be working in without risking their safety or the effectiveness of the system being modelled.
- To conduct safety studies. The graphical or numeric output of a simulation model can be used to identify safety issues. An added benefit is that no one needs to be placed in any danger to acquire the results.

They emphasise that traffic simulation cannot be used as the only tool in the design process and that it should function as a supporting tool in the design process. This makes sense as traffic simulation provides no direct prompts to solutions to traffic problems, but only provides an indication of how well a design is working. It is left to the engineer and other tools in the design the process to address the issues highlighted by a traffic simulation. (Barcelo 2010) identifies three levels of application within traffic simulation:

- Macroscopic traffic modelling
- Mesoscopic traffic modelling
- Microscopic modelling

Macroscopic modelling

Macroscopic traffic modelling is based on the relationships between the variables: speed $u(x,t)$, density $k(x,t)$ and volume $q(x,t)$ as per continuum traffic flow theory. The units of these variables are respectively:

$$q : \frac{number\ of\ vehicles}{time\ elapsed\ (seconds)} \tag{2.1}$$

$$u : \frac{distance\ measured\ over\ (metres)}{time\ elapsed\ (seconds)} \tag{2.2}$$

$$k : \frac{number\ of\ vehicles}{distance\ measured\ over\ (metres)} \tag{2.3}$$

The variable x represents a specific point in space and the variable t represents a specific point in time. The equation that represents this theory can be seen in 2.4

$$\frac{\partial q}{\partial x} + \frac{\partial k}{\partial t} = 0 \tag{2.4}$$

This equation is known as the conservation equation and mathematically represents the assumption that the number of vehicles between two counting stations on a section of road will stay the same for the case that there are no ways for vehicles to enter or leave the road section between the stations.

In addition, the fundamental relationship between u, k and q can be seen in equation 2.5:

$$q(x,t) = u(x,t) * k(x,t) \tag{2.5}$$

Equation 2.3 can be improved by including a term that accounts for vehicles that enter or leave the road section. This is shown in equation 2.6 where the term $g(x,t)$ represents the additional number of vehicles generated on the road, which may be negative.

$$\frac{\partial q}{\partial x} + \frac{\partial k}{\partial t} = g(x,t) \tag{2.6}$$

To find a solution to the conservation equation, another equation is needed that relates either flow and density or equivalently, speed and density. Most commonly, this is either done by using the Greenshields model or the Greenberg model (Garber and Hoel 2015). The Greenshields model assumes that speed and density are linearly proportional. This is expressed in equation 2.7:

$$\bar{u}_s = u_f * (1 - \frac{k}{k_j}) \tag{2.7}$$

Where the terms:

- u_s refers to the space mean speed on the road section
- u_f refers to the average free-flow speed on the road section
- k refers to the car density on the road section
- k_j refers to the density at which traffic on the road section comes to a standstill

By using the fundamental relationship between u, k and q equations for the relationships between flow and speed as well as for flow and density can be developed. For the relationship between flow and speed, the density term k is replaced by the substitution q/u. This leads to equation 2.8:

$$\bar{u}_s^2 = u_f \bar{u}_s - \frac{u_f q}{k_j} \tag{2.8}$$

Equation 2.8 indicates a parabolic relationship between speed and flow. It can be shown that by differentiating q with respect to u and realising that for maximum flow $dq/du = 0$, that the speed at which maximum flow can be expected is half of the mean free flow speed u_f.This result can be found at the turning point of the parabola obtained.

By substituting the speed term $\bar{u}_s$ with q/k we obtain an equation for the relationship between flow and density, shown in equation 2.9:

$$q = u_f k - \frac{u_f k^2}{k_j} \tag{2.9}$$

A parabolic relationship therefore also exists between flow and density. By differentiating q with respect to k and realising that $dq/dk = 0$ for maximum flow, it is shown that the density at which maximum flow can be expected is half of the jam density k_j.

A formula for maximum flow can then be obtained as shown in equation 2.10:

$$q_{max} = \frac{u_f}{2} * \frac{k_j}{2} = \frac{u_f k_j}{4} \tag{2.10}$$

For the Greenberg model, the analogy of fluid flow was used to develop the relationship shown in equation 2.11:

$$\bar{u}_s = c \ln \frac{k_j}{k} \tag{2.11}$$

Where the term c is the speed at maximum flow.

The Greenshields model satisfies the boundary conditions of equation 2.3 when the density k approaches 0 as well as when the density approaches the jam density k_j, while the Greenberg model does not hold up when the density k approaches 0. The Greenshields model can therefore be used for both light and heavy traffic conditions, and the Greenberg model only for dense traffic conditions (ibid.).

Microscopic modelling

Microscopic modelling is an approach that shifts the focus that macroscopic modelling places on road sections to individual vehicles in traffic. Each vehicle's response to the conditions around it, including acceleration, deceleration and turning is modelled. A general equation to represent these responses was developed by General Motors in the 1950s and can be seen in equation 2.12:

$$Response(t+T) = Sensitivity * Stimulus(t) \tag{2.12}$$

Where t represents the simulation timestep and T refers to the reaction time of a driver. An important model of this form for traffic simulation is a car following model which describes the acceleration or deceleration response drivers have to the stimulus of a car ahead of them. The stimulus can be further described as the change in their relative speeds. If it is assumed that the acceleration response is directly proportional to the change in their relative speeds, the linear car-following model is derived. This model is shown in equation 2.13:

$$\ddot{x}_{n+1}(t+T) = \lambda(\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.13}$$

Where the terms:

- $\ddot{x}_{n+1}(t+T)$ is the second derivative of the position of the following car or acceleration response at time $t+T$
- λ is the sensitivity constant
- $\dot{x}_n(t)$ is the first derivative of the leading vehicle's position and therefore is the velocity of the leading vehicle at time t

- $\dot{x}_{n+1}(t)$ is the first derivative of the following vehicle's position and therefore is the velocity of the following vehicle at time t
- The term $\dot{x}_n(t) - \dot{x}_{n+1}(t)$ therefore represents the difference between the velocities of the leading and following vehicle at time t, which is the stimulus to which the following vehicle responds.

(Gazis, Herman, and Potts 1959) suggested that λ be inversely proportional to headway as shown in equation 2.14:

$$\lambda = \frac{c}{x_n(t) - x_{n+1}(t)} \tag{2.14}$$

Equation 2.13 then becomes:

$$\ddot{x}_{n+1}(t+T) = \frac{c}{x_n(t) - x_{n+1}(t)}(\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.15}$$

If equation 2.15 is integrated, the relationship between speed and density becomes as proposed by Greenberg in equation 2.11.

The constant c is then, as for macroscopic modelling, equal to the speed at maximum flow, resulting in 2.16:

$$\ddot{x}_{n+1}(t+T) = \frac{u_m}{x_n(t) - x_{n+1}(t)}(\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.16}$$

(Edie 1961) suggested that Greenberg's model could be slightly improved by assuming that drivers react more sensitively at higher speeds as well as when they were closer to the leading vehicle. Eddie assumed that the sensitivity of the following vehicle was inversely proportional to the square of the headway between the leading and following vehicle. The mathematical consequences of this assumption are shown in equation 2.17:

$$\ddot{x}_{n+1}(t+T) = \frac{constant * \dot{x}_{n+1}(t)}{(x_n(t) - x_{n+1}(t))^2} * (\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.17}$$

Further variability was built into the model by (Gazis, Herman, and Rothery 1961) who proposed equation 2.18:

$$\ddot{x}_{n+1}(t+T) = \frac{constant * (\dot{x}_{n+1}(t))^m}{(x_n(t) - x_{n+1}(t))^l} * (\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.18}$$

Empirical research by (May and Keller 1967) found m=1 and l=3 to be the most suitable values for the variables introduced by (Gazis, Herman, and Rothery 1961). If non-integer values are allowed, improved estimates for m and l can be found. Namely m=0.8 and l=2.8. A further generalised version of the

model was proposed by (Ahmed 1999).
An alternative to assuming that drivers react to the changes in the relative velocity of a leading vehicle, is to assume that drivers place themselves at a distance $\Delta(t)$ behind the leading vehicle such that they are able to stop without crashing into the leading vehicle if the leading vehicle comes to a sudden stop. Making this assumption leads to 2.19 as developed by (Gerlough and Huber 1975):

$$\ddot{x}_{n+1}(t+T) = \frac{1}{T}(\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.19}$$

(Parker 1996) and (Chen et al. 1995) however, found that the assumption that drivers follow at a safe distance is not representative of reality. This is because drivers usually leave a shorter distance than the safe distance between them and the car in front of them. This is represented in equation 2.20:

$$x_n(t+T) - x_{n+1}(t+T) = D_{n+1}(t+T) \tag{2.20}$$

Where $D_{n+1}(t+T)$ is the desired spacing at a time t plus a lag time T. They then postulate that the desired spacing is linearly related to speed as shown in equation 2.21:

$$D_{n+1}(t+T) = \alpha\dot{x}_{n+1}(t+T) + \beta \tag{2.21}$$

Where α and β are model constants. This approach has the advantage that it does not rely on estimates of drivers' reaction behaviour. (Hidas 1998) then showed that if the accelerations of the leading and following vehicles are assumed to be constant, that the acceleration of the following vehicle $a_{n+1}$is given by equation 2.22:

$$a_{n+1} = \frac{1}{\alpha T + 0.5T^2}(T(\dot{x}_n(t) - \dot{x}_{n+1}(t)) + x_n(t) - x_{n+1}(t) - \alpha\dot{x}_{n+1}(t) - \beta + 0.5T^2 a_n) \tag{2.22}$$

T, the time lag can be determined as per equation 2.23:

$$T = \frac{x_n(t) - x_{n+1}(t) - \alpha\dot{x}_{n+1}(t) - \beta}{\alpha a_n - 0.5(\dot{x}_n(t) - \dot{x}_{n+1}(t))} \tag{2.23}$$

(Wiedemann 1974) derived what are called "psycho-physical" car following models. These models are based on two main assumptions:

- When two cars are far apart, the following driver will not be influenced by the size of the difference of the speed of the two cars.

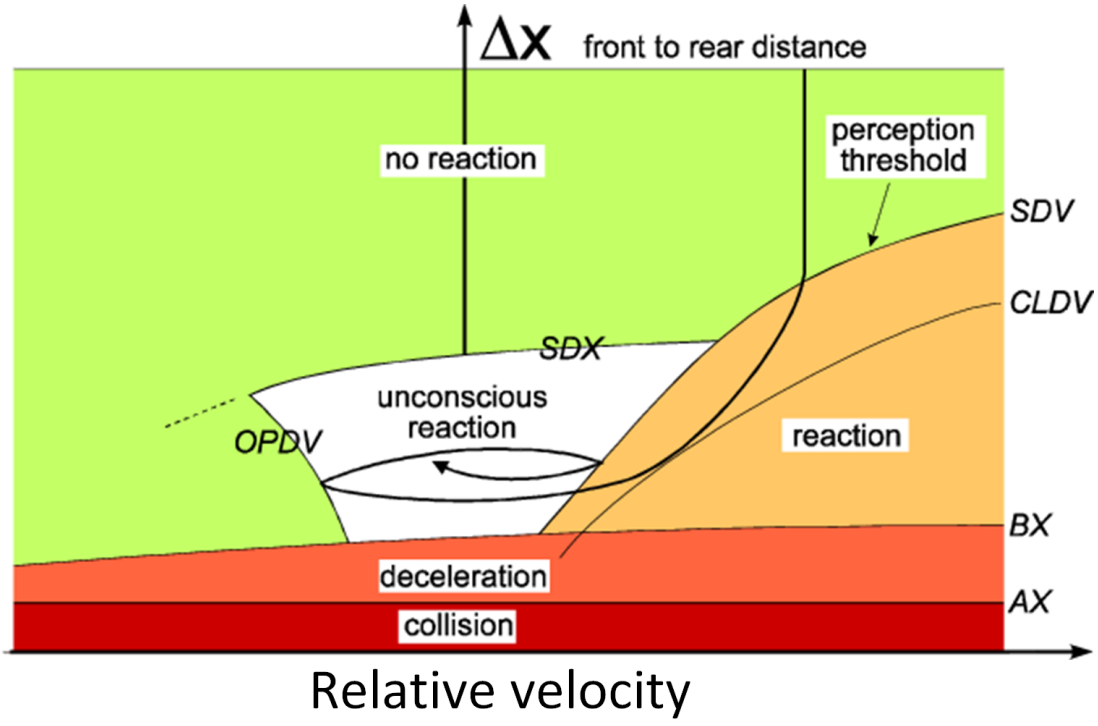


Figure 2.2: Diagram of Wiedemann car following model (Fellendorf and Vortisch 2001)

- When two cars are near, there are certain relative velocities and relative displacements that will not yield a response from the following driver because the relative motion between the two vehicles is negligible.

The following driver will only react when the stimulus provided by the leading car reaches a certain threshold. This is because drivers will not be able to notice small differences in relative speeds. In addition, drivers cannot control their speed perfectly. Because the reactions of the following car will not be perfect, the distance between the cars will increase back to a distance greater than the desired spacing. After this, the following driver will again accelerate to reach the desired spacing. They will then overshoot this distance and then attempt to increase the distance between the vehicles again. The successive deceleration and acceleration of the following vehicle becomes a perpetual cycle. (Leutzbach 1988) showed that this behaviour is a case of 2.18 where m = 0 and l = 2:

$$\ddot{x}_{n+1}(t+T) = \frac{constant}{(x_n(t) - x_{n+1}(t))^2}(\dot{x}_n(t) - \dot{x}_{n+1}(t)) \tag{2.24}$$

This behaviour is shown figure 2.2:
Where:

- AX refers to the desired headway in a standing queue.
- BX is a safety distance that increases with speed.
- SDX is the threshold that models the maximum following distance.
- SDV is the threshold that represents when the driver becomes aware they are approaching a leading vehicle.
- CLDV is the threshold at which a following driver will react by applying their brakes, leading to additional deceleration.
- OPDV is the threshold where a following driver realises that they are slower than the leading car and starts to accelerate again.

The Wiedemann car following model is used in the microsimulation package VISSIM. A final car following model considered, is the Newell car-following model. This model was proposed by (Newell 2002) and is based on the idea that a following vehicle will keep a minimum headway and time gap between it and the leading vehicle. In addition, it is also assumed that each driver in the traffic flow will have a target speed V_n. The acceleration of the following car $a_{n+1}(t)$ is then as shown in equation 2.25:

$$a_{n+1}(t) = \frac{1}{T_{n+1}}\left(\frac{1}{\tau_{n+1}}(x_n(t) - x_{n+1}(t)) - \frac{d_{n+1}}{\tau_{n+1}} - v_{n+1}(t)\right) \tag{2.25}$$

Where:

- a is the acceleration of the following vehicle.
- τ is the time displacement between the leading and following vehicles.
- d is the space displacement between the leading and following vehicles.
- T is the response lag time.
- v is the velocity of the vehicle.
- x is the position of the vehicle.

The mesoscopic modelling approach is a compromise between the macroscopic and the microscopic approach to modelling. This means that some detailed aspects of the microscopic approach are lost. The next section will look at the current state of research that uses simulation for taxi operations in South Africa.

State of research using simulation modelling for taxi operations in South Africa

The first research into the simulation of South African taxi operations took place in 2009 at the University of Pretoria. In their paper, (Fourie 2009) identified shortcomings in the Four Step Method which is the most commonly used technique to model transport demand planning. Specifically, they singled out the Four Step Method's inability to model the dynamic decision-making behaviour associated with commuters. This shortcoming prevents the study of the effects of transport policy changes and infrastructure changes on individual commuters. They proposed that emerging modern technologies such as Agent-Based Simulations be used instead of the Four Step Method. To support this proposal, they reason that Agent-Based Simulation, with its focus on the disaggregated individual role-players in the transportation system, will open opportunities to study such effects and the interactions of individual agents that lead to the observed system-wide behaviour. They further argue that the system-wide behaviour of the minibus taxi industry is clearly a result of the interactions of the various role-players in it and therefore that Agent-Based Simulations are the ideal mechanism to model the industry. (Fourie 2010) conducted a performance comparison of an Agent-Based Simulation framework called MATSim and a Four Step Model called EMME/2. The output from both models was compared to data collected in the real world. They found that while both models produce similar traffic volumes on high-capacity, main routes, MATSim produced travel time distributions that were "dramatically" closer to the observed data. Specifically, they performed a two-sample Kolmogorov-Smirnov test and determined test statistics of 0.6474 for EMME/2 and 0.1784 for MATSim, where a test statistic of 0 indicates perfect agreement with the observed data. (Van Der Merwe 2011) expanded upon the work done by (Fourie 2009) by developing a methodology that could be implemented in the case that limited data was available to calibrate the Agent-Based Simulation model. They also identified numerous opportunities for further research, including:

- Improving the modelling of agent behaviour. In the study, behaviours were determined by assigning a set list of attributes to an agent. It was suggested that the behaviours should rather be based on a model of human intuition, which would give rise to the complex behaviours desired.

- Making it possible for agents to have multiple plans. In the study, each agent had one, fixed plan. It would be more realistic to allow the agents to adapt and improve their plans as the simulation progresses, as do people in reality.

- Improving the distribution and types of activity chains that agents plan. The study used information from a survey in Switzerland to determine

the types and distributions of activity chains encountered in a population. A similar survey in South Africa would produce more appropriate distributions.

(Wevell 2011) made a strong case for the need to simulate transit operations in South Africa. He placed specific emphasis on the minibus taxi industry by highlighting that 50% of commuters in South Africa make use of public transport. These commuters are usually low-income workers and live in underdeveloped, difficult to access areas. He highlights that budget constraints in our country and such areas force planners to develop especially effective and efficient solutions. He champions Agent-Based Simulations as an effective tool in delivering accurate results in this regard. (van-Biljon and CJ Venter 2013) conducted an investigation into the National Department of Transport guidelines for the design of minibus taxi loading facilities or taxi ranks. They identified the following shortcomings in the guidelines:

- The guidelines specify that there must be one berth at a rank for every three taxis in the fleet. This ratio is based on the assumption that for every taxi at the rank, there is one coming to the rank to take its place and one that left the taxi rank from that berth. This approach can only be used if the number of taxis in the fleet is known. Unfortunately, the size of the fleet is unknown when new ranks are built. The size of the taxi fleet is not even certain for existing taxi ranks. In addition, the current size of the fleet may not even be an appropriate indicator for the demand at a taxi rank.
- The ratio method relies on the assumption that routes are of similar lengths, which is not the case in reality.
- The guidelines assume that taxis arrive at the rank in uniform intervals. This is not a valid assumption as it does not hold up with observations made at taxi ranks.
- The guidelines also assume that taxis depart from the rank at uniform intervals. This is not a valid assumption as it does not hold up with observations made at taxi ranks.
- The layout of the taxi rank is also not considered. This is a major shortcoming as it is one of the main factors that engineers have under their control and it has a major effect on the operation of any taxi rank.
- The guidelines only make provision for "small" and "large" urban environments, while the reality is that urban environments can encompass a large variety of sizes. The labels "small" and "large" are also not defined and will be interpreted by the subjective opinions of designers. This will lead to inconsistencies with regard to the results which different designers produce.

To address these shortcomings, (van-Biljon and CJ Venter 2013) developed a simulation method to more accurately determine berth capacities and efficiency. Their motivations for the use of a simulation method are listed below:

- It offers a way of measuring the efficiency and effectiveness of a complex system like a taxi rank where many role-players or agents interact.
- Simulation allows researchers to make observations about processes that occur over a long time in a short amount of time. This is because simulations can be sped up so that time passes faster than in reality.
- Simulations can be used to test hypotheses about existing and imagined systems without having to spend resources creating the test scenarios in reality.

The ability to test such hypotheses is very useful when developing design concepts. Using simulation modelling, they were able to create improved guidelines for the selection of the number of loading berths at a taxi rank as well as the layout that should be used. They conclude their study by making the following recommendations:

- Most importantly, further research into the behaviour of agents at taxi rank is needed.
- By using simulation modelling, the guidelines can be expanded upon.
- The results of such simulation modelling should be presented to the authorities so that new designs can incorporate the insights gained.
- Research into the experience of commuters in terms of safety and convenience is required.

(Neumann, Röder, and Johan W Joubert 2015) applied Agent-Based simulation to a case study on the minibus taxi industry in Port Elizabeth. They improved upon previous work by allowing each agent in the simulation to improve upon its plans by making use of theory related to co-evolution and evolutionary game theory. They claim that they were able to create "close-to-reality" minibus networks. Most research into simulation for taxis was found to take an Agent-Based macroscopic approach. For a taxi rank, a macroscopic approach is not applicable. The lessons learned from applying Agent-Based concepts can however be imaged onto a microscopic taxi rank simulation. For this research project, knowing how the different role-players, which are the agents at a taxi rank, can interact, as well as how they plan their interactions is of importance. Having a solid grasp of these processes will complement the accuracy of the simulation. To gain knowledge about how the agents at taxi ranks interact, observations at taxi ranks need to be made. In addition to this project's own observations, literature containing observations for various taxi

ranks in South Africa was collected. To gain knowledge about how to simulate the plans for interactions that the agents at taxi ranks make, a comprehensive textbook "Automated Planning and Acting" (Ghallab, Nau, and Traverso 2016) was consulted.

Deliberative agents in simulation

(ibid.) define the purpose of planning as coming up with an "organised set of actions" that allow an agent to complete a predefined activity. It can be said that completing the activity is the goal of the agent. The set of actions that any agent decides upon will determine how they conduct themselves in the simulation as well as how they interact with other agents in the simulation. To come up with a feasible or an optimal set of actions, a certain degree of reasoning is required by the agent. The process of reasoning about and deciding on a set of actions for the agent to undertake is called deliberation. The prowess of the human mind as a deliberation entity is well researched and documented. The following attempts to create artificial emulations of its deliberation ability are listed by (ibid.):

- Deliberation with deterministic models. This approach is also called classical planning and makes the following assumptions:
  - That only the agent's actions can change the simulation environment. This means that changes brought about by other agents or any other events are not considered.
  - That the decision-making process does not consider time. This means that it does not make provision for when actions should and can be started, how long they take or how and if it is possible to do several actions at the same time.
  - That the effects of actions are known with certainty. This means that accidents or the chance that actions will fail is not modelled.
  - If these assumptions do not hold for the environment to be simulated, the planning model will produce sub-optimal plans. (ibid.) however, state that this may be acceptable if the errors produced do not have major consequences for the simulation.
- Deliberation with refinement methods. The main idea behind these methods is to break up abstract higher-level tasks into smaller actions that the agents can do. For example, the task "Walk to a bench" could be broken up into the actions: turn toward the nearest bench, then take steps toward the bench until finally, you reach the bench. Breaking up tasks into such a granular level allow agents to have a better idea of what the effects of their actions will be as the effects of these smaller actions are easier to predict. The assumptions made for this kind of planning are:

  - The agents will need to consider the effects of other agents on the simulation environment.
  - The agents will not have perfect knowledge of the state of the entire simulation environment.
  - That actions have a duration that needs to be considered and may happen at the same time as other actions.
  - That actions could have more than one outcome in terms of how they affect the simulation environment.
  - That agents will have to be able to manage discrete and continuous variables.

- Deliberation with temporal models. In this kind of model, the passage of time is explicitly defined. This means that actions, events and happenings in the model do not just have a duration but can also be assigned time-stamps. Knowing exactly when something should or did happen, allows actions and events to be synchronised. Having the passage of time explicitly defined, also allows the modelling of absolute deadlines for goals or actions as well provide the possibility of creating events that occur at a certain time-step in the simulation.

- Deliberation with non-deterministic models. In non-deterministic models, the assumption that the effects of actions are known with certainty is not made. The repercussion of this is that the planning problems that agents are faced with, become much harder to solve. This is because any action can have any number of effects on the simulation environment, just as actions in the real world can have many, variable or unintended consequences. This makes non-deterministic modelling closer to reality than deterministic modelling if done correctly. Because many effects may occur per action in the simulation, agents have a much larger number of possible planning paths to explore. It may not be possible to explore all the possible plan paths. To remedy this, it is recommended that "online planning" is used. Online planning refers to giving agents the ability to change and adapt their plans as the effects of their actions and other events in the simulation become apparent.

- Deliberation with probabilistic models. These models incorporate probabilistic concepts such as Markov chains. This makes it possible for agents to judge the probabilities of the possible planning paths yielding the desired goal. The agent can then factor in these probabilities when choosing a plan.

- Deliberation incorporating:

  - Perception: Where agents make observations throughout the simulation period to keep their knowledge of their environment up to date.

- Monitoring and reasoning: Where agents track their progression in terms of the plans they set out. Doing this may bring about insights about how successful or relevant their plans are and whether alternatives should be considered.
- Learning: Machine learning techniques may be incorporated so that agents can use their experience or knowledge gained from previous simulation runs to make better decisions.
- Ontologies: This is a way to represent and model the classes, hierarchies, relationships and consistency of agent definitions. This kind of description is a powerful technique for establishing how actions or events change the state of the simulation environment.

The formal definition for the planning domain is given in the quadruple 2.26:

$$\Sigma = (S, A, \gamma, c) \tag{2.26}$$

Where:

- Σ is the planning domain. It is the space in which agents are to conduct their deliberation about plans.
- S is the set of states that the simulation environment can be in.
- A is the set of actions that an agent can do.
- γ is the state-transition function. It defines how actions can bring about changes in states.
- c is the cost function. This is the parameter that needs to be minimised in the ideal plan. Examples of costs are time, resource use or monetary expense.

2.5 Conclusion and consolidation of literature review

The literature review highlights that the field being studied is highly dynamic. Any attempt at finding solutions for the field must therefore be targeted and well thought through. The contents discussed in the literature bring the following conclusions:

- A simulation model incorporating deliberative agents may help address the problems statement as discussed in chapter 1.
- A deterministic deliberation model seems most appropriate and achievable.

- Equation 2.15 seems to be the most appropriate car following model to be used.

The next chapters therefore set out to create and validate a simulation model incorporating deliberative agents that can help provide answers to "what if?" questions that can be posed for the study field.

Chapter 3

System design

This chapter provides a broad description of the system that is to be implemented. The system is to be a simulation model that incorporates deliberative agents.

3.1 Overview

The system is a simulation world that models objects in space, the laws of physics and the passage of time. In this world, three distinct types of objects are created:

1. Static objects - These are the objects that are not expected to move during the simulation. They include the three dimensional geometric model of a taxi rank as well as a ground plane or "Terrain" on which the simulation can take place.

2. Dynamic objects - These objects are expected to move during the simulation. They include the agents instantiated during the simulation.

3. Observation objects - These objects are used to make observations and measurements of the simulation world as time progresses in it. They include visual output and data loggers.

The combination of these objects and the simulation world is dubbed the Agent Centred Taxi-Transit Simulation or ACTS.

3.2 Agent identification

The first step was to identify the different role-players at a taxi rank to be included in ACTS. To this end, existing literature was consulted. The different activities that each role-player performs at the loading facility were identified

and the goals that the agents should have were also identified. The two major role-players identified were:

- Taxis
- Commuters

3.3 Agent modelling

Once the agents were identified, they were modelled. The approach used was a combination of finite state machines (FSM) (Russell and Norvig 2009) and goal-oriented action planning (GOAP) (Orkin 2006) to model the decisions taken by agents in ACTS. GOAP is an implementation of a deterministic deliberative planner as described in section 2.4. A complete GOAP model consists of the following elements:

- The GOAP Planner
- The GOAP Agent, including logic for deciding upon the goals of the agent in the simulation.
- The set of actions that an agent can perform. Each action must have defined preconditions and effects.
- The logic for how the agent can interact with the simulation world

The A* (Hart, Nilsson, and Raphael 1968) algorithm was used to model the navigational planning of commuter agents in ACTS. Collision avoidance algorithms were incorporated for each commuter agent. To aid the navigation of the taxi agents, a directed graph that represents the navigable space was created. The taxis use Dijkstra's algorithm (Dijkstra 1959) to choose appropriate paths in the graph. The programming paradigm used throughout the modelling process is object-oriented programming (OOP). The following sub-sections describe the implemented models in more detail.

Finite State Machine implemented

Every agent in ACTS can be in one of three states at a time:

- "Idle" In this state, the agent deliberates over what to do in the simulation by using its GOAP planner component.
- "Move to" In this state, the agent is moving to a target identified by the GOAP planner
- "Perform action" In this state, the agent is performing an action as prompted by the GOAP planner

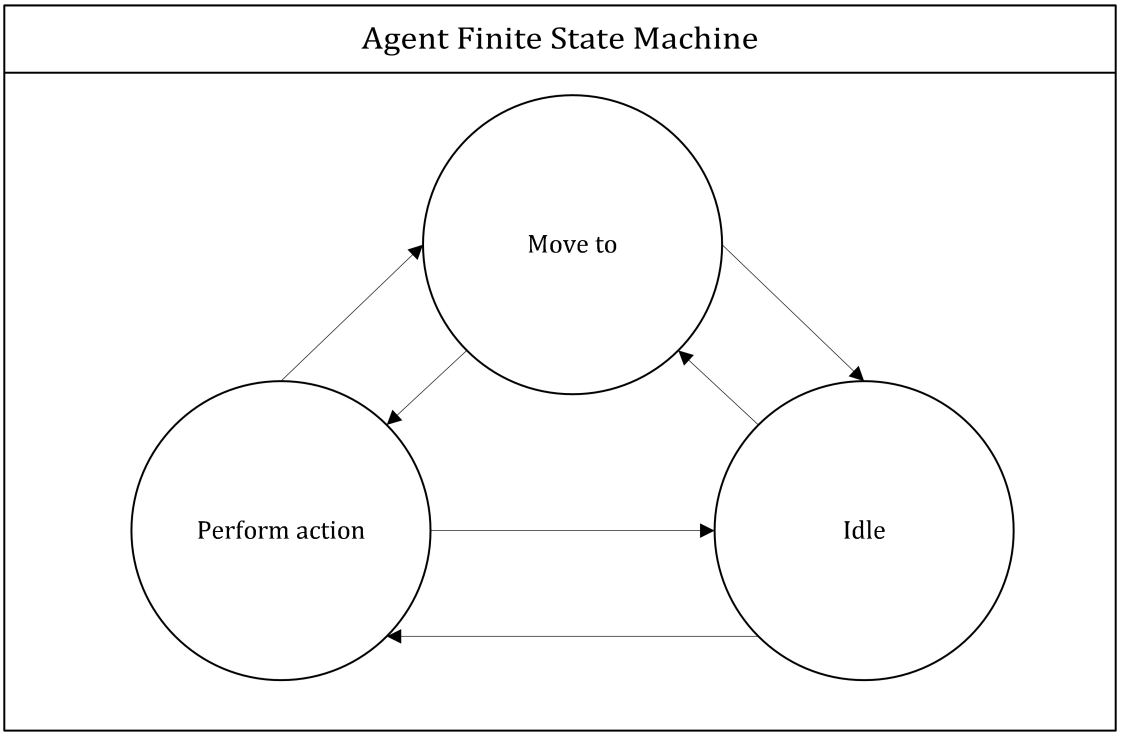


Figure 3.1: Agent Finite State Machine

The agent can transition from any state to another as prompted by the agent's GOAP planner. The FSM is shown diagrammatically in figure 3.1

Goal-Oriented Action Planner implemented

The GOAP planner is what gives agents in ACTS their deliberative power. It allows agents to reason about what to do next in the simulation. The agents observe the current state of the simulation world, consider the set of possible actions made available to them and decide upon a sequence of actions that would best bring about the change to the simulation world that they individually desire, i.e. their goals. A graph of all possible action sequences and their outcomes is constructed and the decision regarding which action sequence to perform is based on choosing the path in the graph that minimises the total cost to achieve the goals of the agent. The GOAP planner as implemented in ACTS can be found in appendix C.

3.4 Static object modelling

The static objects define the constraints placed on the movement of agents by their physical surroundings. The environment that the agents operate in is

a 3-dimensional model. The environment also supports collision detection to prevent agents from displaying unrealistic behaviour like moving through walls or each other. CAD software was used to create 3D models of the structures found at taxi ranks such as splitting islands and office buildings. These models were incorporated in ACTS.

3.5 ACTS model diagram

The relationships between the components of the ACTS model can be seen in appendix B.

The system developed in this chapter was used to inform the development of an implementation of ACTS as described in the chapter that follows.

Chapter 4

Detail design

Bergzicht taxi rank in Stellenbosch Central was modelled in ACTS. This chapter describes the specifics of how the system was implemented for this research project. Topics to be discussed include:

- The software development environment
- The data used as input
- The CAD software made use of
- The open-source libraries made use of
- The key classes developed
- The important algorithms developed

4.1 Development environment

To implement the system as described in chapter 3, it was needed to select a development environment that met the following requirements:

- The environment must be able to simulate physics. This includes modelling the passage of time, the effects of physical forces between the entities in the simulation such as momentum transfers during collisions and gravitational acceleration.
- The environment must support Object Oriented Programming (OOP)
- The environment must be able to incorporate geometric models
- The environment must be able to provide visual as well as data feedback
- The environment must ideally be able to produce portable executable files that run the simulation

- The environment should ideally include a modern feature set and active development community that will enrich the quality of the system implementation
- The environment should ideally have integrated debugging tools

It was identified that the Unity game engine meets requirements. It was therefore used as the main development environment to implement the system. In Unity, any object in the simulation world is called a "GameObject". These GameObjects serve as containers to which "Components" can be added. Examples of components include:

- "Transform" which is attached to any GameObject created. A "Transform" stores a GameObject's location as a vector, its rotation as a quaternion and its scale as a vector.
- "Mesh Renderer"s which allow three-dimensional geometry stored in memory as a "mesh" to be rendered to the screen. This type of component was used to display the taxi rank geometry as well as the visualisations of agents in ACTS.
- "Material" which allows the surfaces of geometry rendered by the GameObject's mesh renderer to appear as realistic textured surfaces. Examples include mimicking the appearance of bricks, paving blocks, paint and other surfaces that can be seen in reality.
- "Collider"s which allow the simulation to know the physical bounds of GameObjects to prevent them from intersecting and passing through each other.
- "Rigid Body" which allows a GameObject to be simulated with the theories developed in rigid body physics.
- "Nav Mesh Surface" which uses the colliders defined for a GameObject to calculate and store what parts of the geometry can be traversed by humanoid agents. In the case of ACTS, the humanoid agents are commuters.
- "Nav Mesh Agent" which allows GameObjects to navigate the surfaces created by the "Nav Mesh Surface" component. To find paths in the surface, the A* algorithm (Hart, Nilsson, and Raphael 1968) is used. It is an adaptation of Dijkstra's algorithm (Dijkstra 1959) that includes a heuristic to prioritise which nodes in a graph, defined as in discrete mathematics, to explore first. The heuristic used is to explore nodes that bring you closer in terms of straight line distance to the end destination first. These algorithms are used to find the shortest path between two nodes that exist in a graph. An example would be to consider a road network as a graph where

the nodes are the intersections and the roads are edges of the graph. One could then Dijkstra's algorithm or A* to find the shortest route between any two intersections. The "Nav Mesh Agent" uses the navigable surfaces produced by the "Nav Mesh Surface" component as nodes and their adjacency as edges. The advantage of using A* is that it may return a result quicker or at least as fast as Dijkstra's algorithm.

- "Script" components. These components allow C♯ classes to be attached to a GameObject. This powerful feature allows the use of OOP models and direct control over the simulation world in the C♯ programming language. Script components are the main vehicle by which behaviours were added to GameObjects.

Additional useful features of GameObjects are that they can be nested and can be saved as "Prefabs". A prefab is an example of a GameObject that others can be cloned and instantiated from. Prefabs created for ACTS include:

- AITaxi - This prefab is used to instantiate taxis into the ACTS simulation world
- Commuter - This prefab is used to instantiate commuters into the ACTS simulation world
- NavNode - This prefab is used to instantiate GameObjects that represent places that taxis navigate. The taxi navigation system is discussed in more detail in section 4.7.
- BayNode - This prefab is used to instantiate GameObjects that represent places that taxis can pick up commuters. The commuter loading system is discussed in more detail in section 4.7.
- ParkingNode - This prefab is used to instantiate GameObjects that represent places that taxis park while waiting for space to be available in the rank aisles. The taxi parking system is discussed in more detail in section 4.7.
- ExitNode - This prefab is used to instantiate GameObjects that represent places that taxi and commuters can leave the rank by. This system logs relevant data about the agent leaving and saves it to an output text file. This system is discussed in more detail in section 4.7.

4.2 Data used as input

For an update on its Comprehensive Integrated Transport Plan (CITP), Stellenbosch Municipality commissioned taxi rank surveys to be done at the following minibus taxi ranks:

- Bergzicht
- Kayamandi
- Klapmuts
- Franschhoek

Surveys (Kantey and Templar Consulting Engineers 2018) were conducted at these ranks for the weekday morning peak time from 06:00 until 09:00, at weekday afternoon peak time from 15:00 until 18:00 as well as for Saturday peak time from 11:00 until 14:00. The results of the survey were compiled into a spreadsheet with the following fields:

- Rank at which survey took place
- Time that the taxi joined the queue
- Number of passengers getting off the taxi
- Time that the first passenger gets on the taxi
- Vehicle registration (number plate)
- Destination of taxi
- Fare
- Route code
- Number of passengers when the taxi departs
- Time that the taxi departs
- Date of the observation

For testing ACTS, the records pertaining to Bergzicht taxi rank were used. The data recorded in the spreadsheet was used as input data to instantiate taxi agents in ACTS. The most important field was the time that the taxi joined the queue. This time was used as an arrival time at the rank. After allowing passengers to disembark and then loading new passengers, taxi in ACTS eventually leaves the taxi rank and the time at which it leaves is recorded. This predicted departure time can be compared to the observed departure time recorded in the spreadsheet. This is a mechanism by which the validity of the results produced by ACTS can be evaluated.

4.3 CAD software made use of

A geometric model of the taxi rank was needed. The closest only entity that contained geometric information identified during the research process was a PDF file showing the rank in plan view. The rank plan can be seen in appendix A.

Upon closer inspection, it was determined that the plan was drawn as vector graphics. Since many CAD programs accept vector graphic files such as Standard Vector Graphics (SVG) files, it was decided that the vectors needed to be extracted and written to an SVG file. For this end, the open-source vector graphics editing software package Inkscape was used. Once the vectors had been written to a SVG file, the file was imported into the open-source 3D computer graphics software toolset Blender. Blender was chosen for its rich feature set, active development community and its output being directly supported by the Unity environment. Blender was used to delete all the unnecessary vectors such as text and detailing. It was also used to extrude the geometry of the rank's parallel islands, walls, bollards and buildings. The extruded geometry was three dimensional and surfaced. In addition, an orthophoto obtained from the United States Geological Survey's LandsatLook online tool was used as a background. The created 3D model was then imported into Unity as a static GameObject and given mesh renderer, mesh collider, nav mesh surface and material components. The rank geometry as it was imported into Unity can be seen in figure 4.1. The light blue colouring represents the nav mesh surface. It can be seen that the nav mesh surface generated excludes areas such as walls and bollards from the navigable space. This is a feature of the nav mesh surface component being able to interpret and use 3D geometry to make logical descisions about what areas of the geometry are walkable and which are not. This is an advantage of the 3D approach taken in this research project.

4.4 Open-source libraries used

This section identifies and discusses open-source libraries used in ACTS.

Unity Standard Assets

The Unity Standard Assets package consists of a variety of geometric models, classes and prefabs that are generally required for many of the projects made in Unity. In ACTS, the following components in the Unity Standard Assets package were used:

Prefab "CarWayPointBased.prefab"

This prefab represents a vehicle with 4 wheels that can be controlled by sending it waypoints to follow.

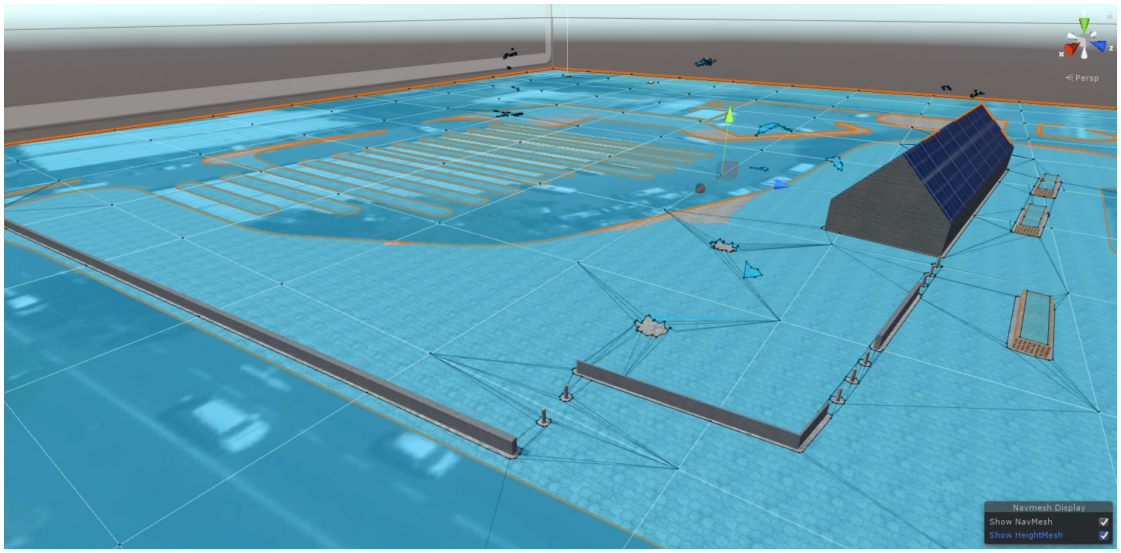


Figure 4.1: Oblique view of rank as imported into Unity

Class "CarController.cs"

This class provides the logic for how a vehicle modelled with CarWayPoint-Based.prefab reacts to accelerator, brakes and steering inputs. Examples of factors controlled by the CarController.cs class are:

- Car drive type, i.e. four wheel drive, rear wheel drive or front wheel drive
- The maximum steer angle for the steering wheels
- The degree of traction control provided to the wheels
- The amount of torque available to the wheels
- The amount of downforce experienced by the vehicle
- The top speed of the vehicle
- The maximum boundary of the vehicle's rev range
- The amount of braking torque available to the vehicle

This use of this class differentiates ACTS from most other traffic simulation software. This is because most traffic simulation software packages directly control the acceleration of their vehicles by the use of the car following models discussed in chapter 2. In ACTS, however the car following models are used to determine how strongly a vehicle should accelerate or decelerate. Equation 4.1 shows the car following model used in ACTS. This information is then used to generate accelerator, braking and steering input. The CarController.cs class

then applies the effects of the input generated to the vehicle. This process more closely resembles reality and provides granular control over the steps in it.

Class "CarAIControl.cs"

This class generates accelerator, braking and steering input. As provided in Unity Standard Assets, it has access to the next waypoint for the vehicle. It then uses the following factors to decide on appropriate inputs:

- How cautious the vehicle should be regarding speed
- How cautious the vehicle should be regarding cornering
- How sensitive the vehicle should be regarding accelerator inputs
- How sensitive the vehicle should be regarding braking inputs
- How sensitive the vehicle should be regarding steering inputs
- How much lateral wander the vehicle should have when driving. This is to mimic the way drivers in reality never drive in a perfectly straight line.
- How much the acceleration input should wander. This mimics the way that drivers in reality do not accelerate uniformly.
- Whether the vehicle should stop at the next waypoint.
- Whether the vehicle uses the distance to the waypoint or direction difference to decide on a braking input or whether it should never brake.

In ACTS the functionality of CarAIControl.cs was extended by creating a method called "Sense()". The purpose of this method was to make the vehicle aware of its surroundings other than the next waypoint. This includes awareness of other vehicles, pedestrians and the physical environment surrounding the vehicle. The vehicle can then use this information to decide on appropriate acceleration or deceleration as per the car following models discussed in chapter 2. The "Sense()" function is discussed in detail in section 4.7.

Goal Oriented Action Planning AI in Unity

This library contains abstract classes and interfaces that facilitate the creation of GOAP models. The following classes and interfaces are provided:

FSMState.cs

This class represents a way to create the enumeration of all the different states that an agent can be in. The states implemented in Goal Oriented Action Planning AI in Unity and therefore ACTS are:

- Idle: A state in which an agent decides what it should do.
- MoveTo: A state in which an agent travels to where it can do what it wants to do.
- PerformAction: A state in which an agent does what it had set out to do.

FSM.cs

This class implements a model of a finite state machine. The data structure used in the implementation is a stack of FSMStates. The reason a stack is used, is to ensure that there is no ambiguity as to what state should be switched to when transitioning from the current state: It is always the next one that is popped off of the stack.

GoapAction.cs

This abstract class provides a definition for modelling actions that agents can do should be modelled after. It consists of the following important attributes:

- private Dictionary<string, object> preconditions - This dictionary stores the preconditions of an action as KeyValuePairs where the keys are name strings and the values are objects that represent conditions regarding the simulation world that need to be met for the agent to be able to do the action.
- private Dictionary<string, object> effects - This dictionary stores the expected effects of an action as KeyValuePairs where the keys are name strings and the values are objects that represent the changes to simulation world state that are expected when the action is completed.

GoapAction.cs also defines the following important methods:

- public abstract void reset() - This method allows a developer implementing a subclass of GoapAction to reset any variables that need to be reset before repeating the GOAP Planning procedure.
- public abstract bool isDone() - This method allows a developer implementing a subclass of GoapAction to define when the action to be implemented can be considered as complete.

- public abstract checkProceduralPrecondition(GameObject agent) - This method allows a developer implementing a subclass of GoapAction to indicate to the GOAP planner whether the action should be considered when making a plan. It is provided for situations where it is not convenient to include a condition in the preconditions dictionary.
- public abstract bool perform() - This method is run when the action is performed. In it, the effects that actions have on the simulation world are to be implemented. It returns whether the action was successfully performed or not.
- public abstract bool requiresInRange() - This method should return true if the action can only be done when in range of a target and should return false when the action can be performed anywhere.
- public bool isInRange() - This action indicates whether the agent is close enough to the target of the action to be able to perform it or not.
- public void setInRange(bool inRange) - This method allows a developer to specify that an agent is in range of the target of its next action when this is true.

GoapPlanner.cs

This class implements a GOAP planner as described in chapter 3. The important method in GoapPlanner.cs is:

public Queue<GoapAction> plan(GameObject agent,
HashSet<GoapAction> availableActions,
Dictionary<string, object> worldState,
Dictionary<string, object> goal)

The method returns a sequence of GoapActions that will allow the goals of the agent to be satisfied.

IGoap.cs

IGoap is an interface that any class that represents a GOAP empowered agent must implement. It forces the implementation of methods that provide information to the GoapPlanner class as well as feedback over whether a plan was completed successfully or not. The methods that must be implemented when using IGoap are:

- Dictionary<string, object> getWorldState() - This method allows the agent to pass what it perceives the current world state to be to the GOAP planner.

- Dictionary<string, object> getGoalState() - This method allows the agent to pass its goals to the GOAP planner.
- void planFailed(Dictionary<string, object> failedGoal) - The GOAP planner will call this method when it cannot construct a valid sequence of GOAP actions that will allow the agent to achieve its goal. The goals that could not be achieved are passed to the agent. The agent can then reason about how to adapt or change its goals before starting the planning process again.
- void planFound(Dictionary<string, object> goal, Queue<GoapAction> actions) - The GOAP planner calls this method when it has found a valid sequence of actions that the agent can perform to achieve its goals. The goals as well as the sequence of actions in the form of a queue are passed to the agent. The agent can then decide to do what is relevant when receiving a valid plan. At this point, it is useful to display what sequence of actions the agent is endeavouring to take on for debugging purposes.

GoapAgent

This class represents the engine that keeps the gears of GOAP turning. It must be attached to any GameObject that is to be a GOAP Agent in the simulation. It initialises the finite state machine, GOAP planner, set of available actions and a queue that will hold the actions that the agent plans to do. It then pushes the idle state to the finite state machine stack so that the agent can start planning what to do in the simulation. Once a plan has been created, the agent pushes moveTo and performAction states as required to fulfill the plan.

The source code was slightly altered for ACTS by replacing all references to "HashSet< KeyValuePair<string, object> >" with "Dictionary<string, object>". Functionally, these definitions are identical, but it was felt that the latter was slightly more readable and workable.

Accord.NET

Accord.NET is a package of libraries that can be used for scientific computing. The classes used were:

- Accord.Math.Random.Generator - This class was used in the algorithm that calculates the inter-arrival times for all commuters that walk into the rank. See section 4.7 for more details regarding the algorithm.
- Accord.Statistics.Distributions.Univariate. LognormalDistribution - This class was used in the algorithm that calcu-

Action name	Preconditions	Effects
Wait in parking	none	Bay available
Alight passenger	Taxi is stopped	Passenger alighted
Load passenger	Bay available Stopped at bay Passengers alighted	Passenger loaded
Leave rank	Passengers loaded	Taxi leaves rank

Table 4.1: Set of actions available to taxi agents

lates how long it should take for a passenger to board a taxi. See section 4.7 for more details regarding the algorithm.

4.5 GOAP model developed

GOAP Model of Taxis

It was decided that the goal a taxi has when using a rank is to leave with a full complement of passengers. This is so that they can maximise their profit per trip. To do this however, they first need to navigate to the bay that corresponds to their destination and unload any passengers that wish to alight. If, however, an appropriate bay is not available, they must first navigate to and wait in a designated parking area. The action set provided to the taxi agents can be seen in table 4.1. The agents can query the world state to find out whether the preconditions of the various actions have been met or not. They can then create a plan that will satisfy their goal with the least cost incurred. For the action set of the taxi agents, the only action that has a cost is "Wait in parking". This is done so that taxi agents will avoid waiting if it is not necessary.

GOAP Model of Commuters

The goal of commuters is to reach their destination. If they arrive at the rank on foot, it means that they want to get on a taxi that is going to their destination. To get on an appropriate taxi then becomes the goal of the agent. If they arrive at the rank via a taxi, it means that they have gone as far as they wish via a taxi and would like to walk out of the rank. To leave the taxi rank on foot then becomes the goal of the agent. They may also want to transfer to a different taxi. This action could be incorporated into the ACTS model if information regarding transfers was available. This action was not included in the action set as only 66 of the 592 taxis observed had any people disembarking. The number of people transferring to another taxi will be at most as much and most likely much less than the number of people disembarking. In addition,

Action name	Preconditions	Effects
Get on taxi	none	Get to desired destination
Leave rank	none	Commuter leaves rank

Table 4.2: Set of actions available to commuter agents

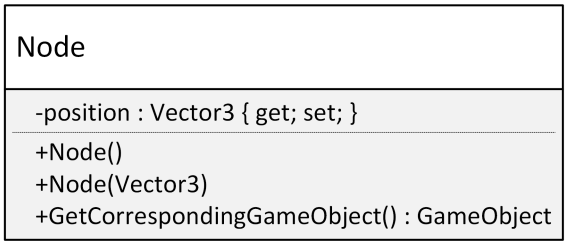


Figure 4.2: Class Node

commuter transfers times do not affect the loading times of taxis. The action set that models commuters in ACTS is shown in table 4.2. ACTS assigns a goal to a commuter depending on their mode of arrival.

4.6 Key classes developed

This section details the important classes developed for ACTS. Their significance, important attributes and important methods are discussed.

Node

This class is used to create objects that represent places that taxis can drive to in the simulation environment. A Node has one major attribute: private Vector3 position. Where the type Vector3 refers to a data structure that stores a vector of length 3. The elements of the vector are the x, y and z cartesian coordinates that represent a position in the simulation. In Unity y is used to represent the upward direction in the simulation world. The coordinates are stored as floating point values. In addition to a getter and a setter for the position, attribute, Node objects also have a method called: public GameObject GetCorrespondingGameObject(). The purpose of this function is to identify which NavNode GameObject in the simulation world corresponds to the Node object. A NavNode GameObject that is at the same position as the Node object is returned by this method. For more information on how the taxi agents in ACTS use Nodes and NavNodes to find their way around the modelled taxi rank can be found in 4.7. A UML diagram of class Node can be seen in figure 4.2.

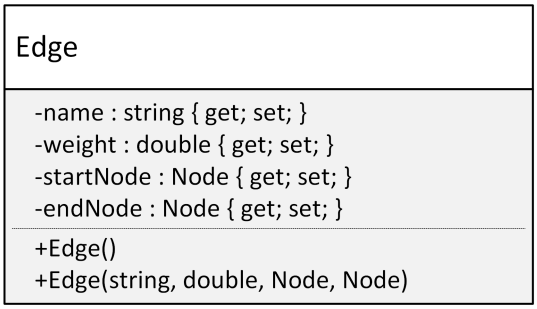


Figure 4.3: Class Edge

Edge

Edge objects represent how Node objects should be connected in the simulation world. They consist of the following attributes:

- private string name - This attribute provides a way of naming and keeping track of Edges.
- private Node startNode - This attribute is the Node that the Edge starts at.
- private Node endNode - This attribute is the Node at which the Edge ends.
- private string weight - This attribute specifies how difficult it is to traverse the Edge. In ACTS the weight is calculated as the straight line distance between the starting Node and ending Node of the Edge.

Getters and setters for the various attributes are also provided in the Edge class. A UML diagram of class Edge can be seen in figure 4.3.

Vector3Graph

This class represents a graph that consists of Nodes connected by Edges. The Nodes are stored as a HashSet of Nodes and the Edges as a HashSet of Edges. The most important method in Vector3Graph is:
public Dictionary<Node, Node> FindShortestPathDijkstra
(Node startNode, Node endNode)
This method is used to find the shortest path from any Node in the graph to any other Node in the graph as calculated by Dijkstra's path-finding algorithm as implemented specially for use with Unity Vector3s. The method returns a Dictionary in which both the keys and values are Nodes. When the Dictionary is asked to return a value associated with a Node as a key, the value it returns is the Node that follows the key Node in the path. The Dictionary can then be asked to return the value associated when using the returned value as a

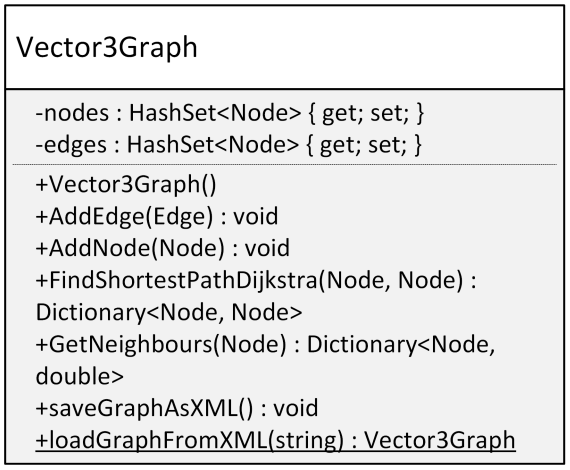


Figure 4.4: Class Vector3Graph

key. This process can be continued until the Dictionary can no longer return a value for the key used. At this point, it is known that the last value returned was the last Node in the path. Another important method in Vector3Graph is Dictionary<Node, double> GetNeighbours(Node node). This method takes a Node as an argument and returns all the Nodes that are direct neighbours of that Node. The return type is a Dictionary where a key is a neighbouring Node and a value is the weight associated with going to that Node. The GetNeighbours(Node node) method only checks Edges that start with the Node given as its argument. This makes the graph directed. It can be made bidirectional by also checking the Edges that end at the argument Node. In ACTS however, it is more convenient to merely create another Edge in the apposing direction to an Edge that should be bidirectional. Class Vector3Graph also provides functionality to save and load serialised versions of its instances. The files created to represent the instances are in eXtensible Markup Language (XML) format. The performance of the algorithms and data structures used in Vector3Graph were found to be acceptable for the ACTS model. A UML diagram of class Vector3Graph can be seen in figure 4.4.

Destination

Class Destination maintains a record of all the different destinations that commuters and taxis want to travel to during the simulation. A Destination has the following attributes:

- private string name - This is what the destination is called. Examples included "Somerset West", "Idas Valley" and "Kayamandi".

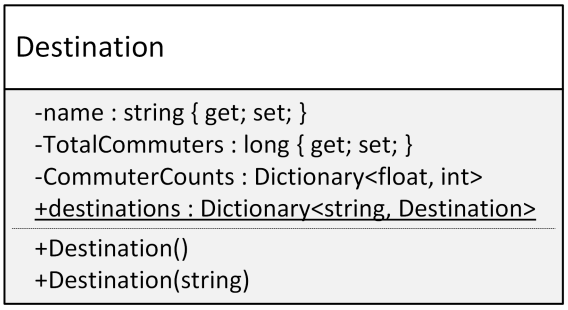


Figure 4.5: Class Destination

- private long TotalCommuters - This number represents the total number of commuters expected to want to travel to the destination during the simulation period.
- public Dictionary<float, int> CommuterCounts - This attribute stores how many commuters are expected to arrive at any one simulation time step. The keys in the Dictionary are floating point numbers that represent the number of seconds that have passed since the beginning of the simulation. The values in the Dictionary represent the number of commuters expected and is stored as an integer. This attribute is used during the calculation of inter-arrival times for all commuters in the simulation. For more details on the algorithm for this calculation, see section 4.7

A static attribute also stores all created instances of Destination in a public static Dictionary<string, Destination> where the keys are the names of the stored destinations and the values are the stored Destinations themselves. A UML diagram of class Destination can be seen in figure 4.5.

CommuterQueue

This class represents the queue of commuters that are waiting to be picked up by a taxi at a bay. It has functionality to add commuters to the back of the queue, remove commuters from the front of the queue and query the last GameObject in the queue for its Transform is. The Transform of the last commuter GameObject in the queue represents the end of the queue to which any commuter wishing to join the queue must walk to. A UML diagram of class CommuterQueue can be seen in figure 4.6.

Bay

A Bay is a representation of where a taxi can stop to load passengers and commuters can queue up while waiting to board taxis. Important attributes of Bay include:

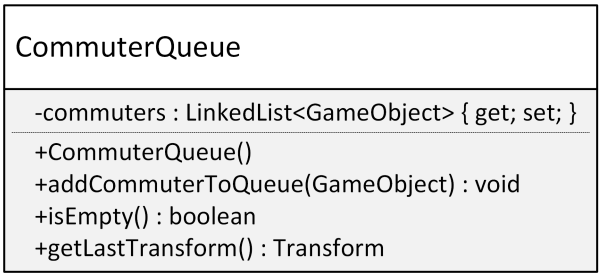


Figure 4.6: Class CommuterQueue

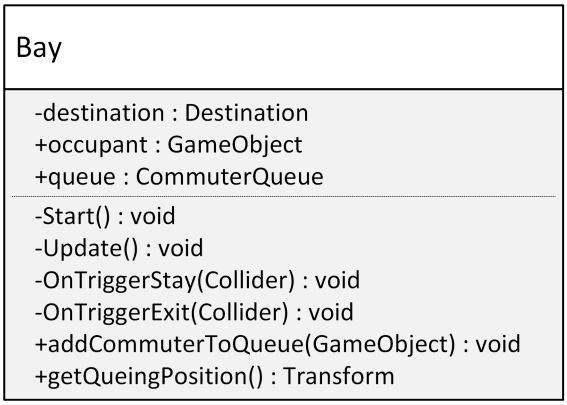


Figure 4.7: Class Bay

- private Destination destination - At the taxi rank studied in this project, taxis must stop at the bays labelled with the name of the destination that they are going to.

- public GameObject occupant - This is the current GameObject that is occupying the Bay.

- public CommuterQueue queue - This represents the queue of commuters waiting to board a taxi at that Bay.

A Bay object also checks whether a taxi that still has space for commuters is parked at it. If the such a taxi is parked at the Bay and there are commuters in the CommuterQueue of the bay, the Bay signals to the taxi that it can load a passenger. For more details on the passenger loading algorithm see section 4.7. The Bay class is attached to any GameObject created from the prefab "BayNode". A UML diagram of class Bay can be seen in figure 4.7.

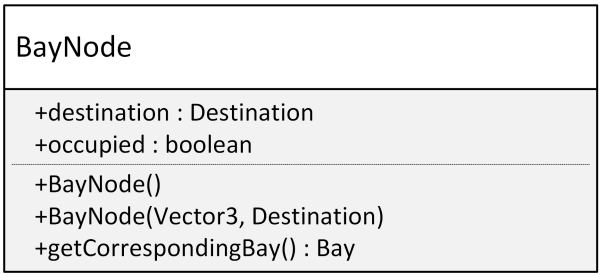


Figure 4.8: Class BayNode

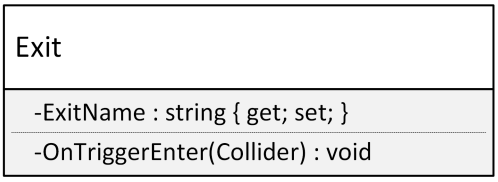


Figure 4.9: Class Exit

BayNode

BayNode is a subclass of Node and represents a Node at which a taxi agent can pick up passengers. In addition to the attributes of Node, it defines the following attributes:

- public Destination destination - The destination associated with the BayNode
- private bool occupied - This variable indicates whether a taxi is parked at the bay or not. It is used by the GOAP Planners of arriving taxis to determine whether they can park at the bay to load passengers or whether they must park elsewhere and wait for the bay to open up before going to load passengers at it.

A UML diagram of class BayNode can be seen in figure 4.8.

Exit

The Exit class is attached to any GameObject created from the "ExitNode" prefab. Class Exit allows the GameObject it is attached to record the time at which a taxi exits the taxi rank. The predicted departure time, time of entry, identifying features, destination and observed departure time for the taxi are recorded and saved to a Comma Separated Values (CSV) file. The taxi GameObject is then destroyed. A UML diagram of class Exit can be seen in figure 4.9.

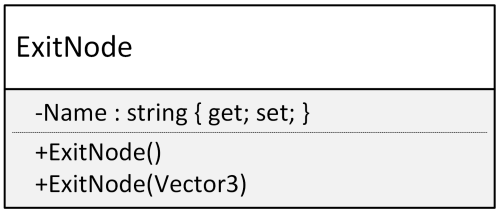


Figure 4.10: Class ExitNode

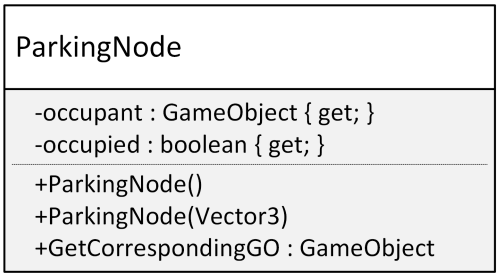


Figure 4.11: Class ParkingNode

ExitNode

ExitNode is a subclass of Node. Its distinguishing feature is that it is where the taxis and commuters can travel to exit the simulation. It is used by the taxi navigation system to find an appropriate exit from the rank. A UML diagram of class ExitNode can be seen in figure 4.10.

ParkingNode

The ParkingNode class is a subclass of Node. It represents a place where taxi agents can stop and wait for a aisle to a Bay to become available. A ParkingNode object can return which "ParkingNode" prefab corresponds to it as well as report whether it is occupied or not. A UML diagram of class ParkingNode can be seen in figure 4.11.

Taxi

Class Taxi is a model of a minibus taxi at a taxi rank. It implements the interface IGoap. It has the following important attributes:

- private Destination nextDestination - The destination to which the taxi will be travelling.
- public string ID - An identifying string to keep track of the taxi during the simualtion.

- public string ExpectedDeparture - A string that stores the time at which the taxi was observed to depart the rank. This value is obtained from the taxi rank survey conducted by Stellenbosch Municipality.
- private float arrivalTime - The time in seconds after the start of the simulation that the taxi is scheduled to arrive. The time recorded by the taxi rank survey is used to calculate this value when initialising a taxi. For more details regarding the initialisation of taxi agents, see section 4.7.
- private string RouteNumber - This string stores the route number that the taxi is registered to service.
- private int maxSeated - This attribute represents the maximum number of passengers that the taxi can load. It is usually set to 15 as per regulation.
- private float fare - This attribute stores the fare that a taxi charges. It was obtained from the taxi rank survey.
- private Node nextNode - This attribute represents the next Node that the taxi is travelling toward as directed by the taxi navigation system.
- private Queue<Node> path - The path attribute is a queue of the nodes that the agent plans to travel via when traversing the rank. When a taxi approaches its nextNode, it dequeues a Node from the path attribute and assigns it to its nextNode attribute. The taxi then continues to the nextNode and repeats the process until there are no more Nodes in the path.
- private Node endNode - This is the last Node in the path. It is used to determine how close the taxi is to the end of the path.
- private GameObject navtarget - This attribute is a GameObject that is used by the CarAIControl class to decide what steering input to pass to the CarController class. The Taxi sets the position of the navtarget to be the same as the position of the nextNode attribute.
- public bool alightingPassenger - This value of this boolean variable reflects whether the Taxi is currently allowing a passenger to alight or not.
- public bool loadingPassenger - This value of this boolean variable reflects whether the Taxi is currently loading a passenger or not.
- public bool alightedPassengers - This value of this boolean variable reflects whether all the passengers that were on the Taxi upon initialisation have alighted or not.
- private LinkedList<GameObject> passengers - This attribute is the collection that contains all the commuters currently aboard the Taxi.

- private Dictionary<string, object> worldState - This attribute stores a representation of information regarding the simulation world that is relevant to the agent decision making process. It stores the relevant aspects of the simulation world as the taxi agent perceives them.
- private Dictionary<string, object> goalState - This attribute stores a representation of the current goals of the taxi agent.

Class Taxi also has the following important methods:

- public Taxi() - A constructor for a Taxi object. An overloaded version of this method that accepts initial values for various attributes as parameters is also provided in class Taxi.
- private void Start() - This method is called whenever a Taxi agent is instantiated. When called, it calls the initializeWorldState() and initializedGoalState() methods described below.
- private void initializeWorldState() - This method creates a new dictionary for the worldState attribute and adds the representation of what the agent perceives to be the world state to the dictionary. For a taxi arriving at the rank, the following world state is created:
  - It is false that the taxi has loaded passengers at the rank.
  - It is false that the passengers on the taxi have alighted.
  - It is false that the taxi has stopped.
  - It is false that the taxi has left the rank with a full complement of new passengers.
  - It is either true or false that there is a Bay available to load passengers at. The taxi queries the simulation world for this information and then adds the appropriate information to the worldState.
- private void initializeGoalState() - Here, the goals of the taxi can be initilised. For the taxi agents in ACTS, the goal of a taxi is to leave the rank with a full complement of passengers.
- public IEnumerator addPassengerFromQueue(CommuterQueue queue) - This method is called to load a passenger on to a Taxi from a CommuterQueue. The CommuterQueue and call to load a passenger is usually provided by a Bay object. This method uses the existing model for passenger loading times developed by (van-Biljon and CJ Venter 2013). For more details regarding the implementation of this method, see section 4.7.
- public IEnumerator alightPassenger() - This method allows a passenger on the taxi to alight.

- private void Update() - This method is called every time a visual output frame is drawn during the simulation. Usually, the simulation draws sixty frames per second, so Update() will usually be called sixty times per second. For class Taxi, the Update() method does two important things:
  - Check whether the taxi agent is near the next navtarget so that the next Node in the path can be dequeued.
  - Check whether the taxi agent is close to the endNode so that it can begin slowing down.
- public void planRoute(Node endNode) - The planRoute method accepts a Node and uses class Vector3Graph to determine whether there is a valid path through the taxi navigation graph to the Node. If such a path exists, the path attribute of the Taxi is set to that path. The endNode attribute of the Taxi is then also set to the Node provided as an argument in the planRoute method. More details regarding the taxi agent navigation system can be found in section 4.7.
- public bool isBayAccesible() - This method is used by a taxi agent to determine whether it can drive to a Bay to load passengers or whether it should go park in a waiting parking area.
- public bool isTaxiFull() - This method checks whether the taxi is holding the maximum number of people it can seat or not.
- public Dictionary<string, object> getWorldState() - This getter method provides external access to the worldState attribute. The IGoap interface requires that this method be implemented.
- public Dictionary<string, object> getGoalState() - This getter method provides external access to the goalState attribute. The IGoap interface requires that this method be implemented.
- public void planFailed(Dictionary<string, object> failedGoal) - The IGoap interface requires that this method be implemented. In ACTS, it is used by taxi agents to log that the plan that they had made had failed at some point during the execution process. The agents then enter the idle state again and re-plan. It is a "back to the drawing board" moment.
- public void planFound(Dictionary<string, object> goal, Queue<GoapAction> actions) - The IGoap interface requires that this method be implemented. In ACTS, it is used by taxi agents to log the action steps that they plan to take to achieve their goals.
- public void actionsFinished() - The IGoap interface requires that this method be implemented. It is called when all the steps in the plan of

an agent have been carried out by that agent. In ACTS, this method is used to log when an agent achieves its goals.

- public void planAborted() - The IGoap interface requires that this method be implemented. It is called by the GoapAgent when an action cannot be completed and the current plan must be abandoned. In ACTS it is used to log when such an event happens for an agent.

- public bool moveAgent(GoapAction nextAction) - This crucial method defines how a taxi agent should move to the position where it can conduct the next action in its action sequence. It works by checking whether the agent is not already navigating to the required position and then calling the planRoute(Node endNode) method if the agent is not already travelling to the required position. This method returns false while the taxi agent has not reached the required position. Once the taxi agent reaches the required position for the next action, this method returns true and signals to the action that it can be performed. The IGoap interface requires that this method be implemented.

- public ParkingNode GetApplicableParking() - This method searches for and returns an unoccupied ParkingNode where the taxi agent can wait for a Bay to become available.

- public GameObject GetAppropriateBay() - This method is used to find an appropriate GameObject with an attached Bay component for the taxi agent to load passengers at. "Appropriate" refers to the requirement that the passengers queuing at the bay and the taxi agent have the same destination in mind.

A UML class diagram of class Taxi can be seen in figure 4.12.

LoadPassengersAction

Class LoadPassengersAction is implemented as a subclass of class GoapAction. It represents the action of loading passengers from a queue of commuters waiting at a loading bay. It has an attribute: private bool passengersLoaded to keep track of whether the action has been completed. LoadPassengersAction also implements the following methods:

- public override string ActionName - This is a getter method and returns the string "LoadPassengersAction". It is inherited from GoapAction.

- public LoadPassengersAction() - This method is the constructor for a LoadPassengersAction object. In it, the following preconditions and effects are described:

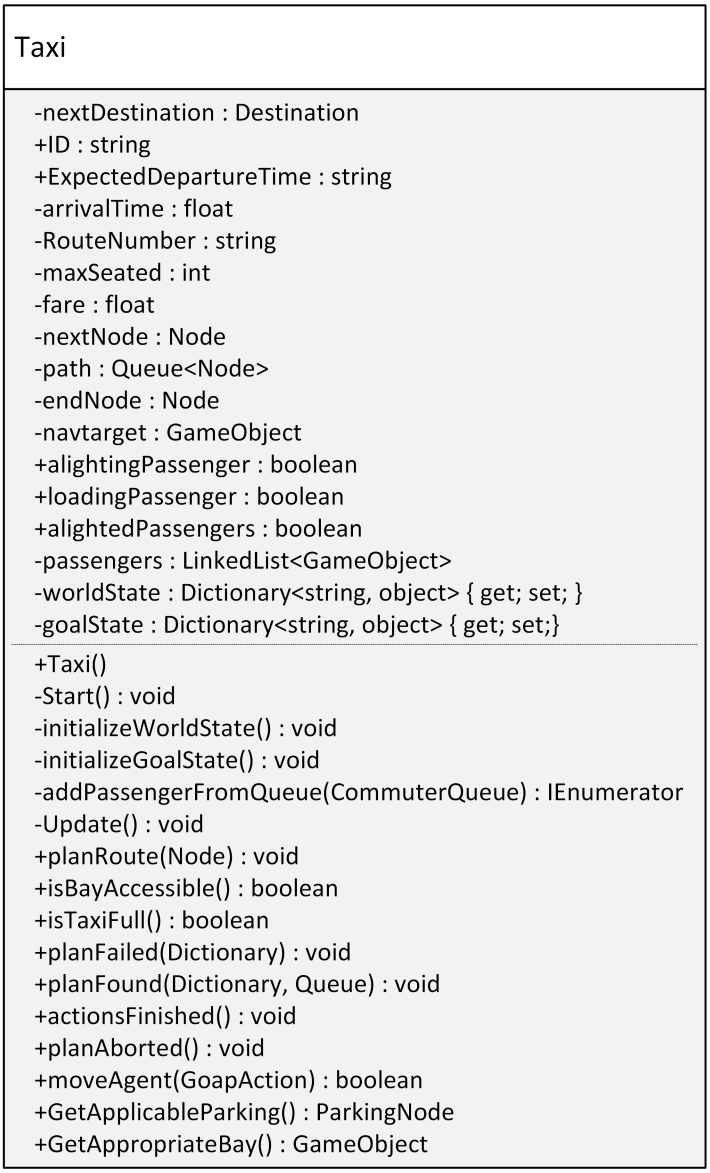


Figure 4.12: Class Taxi

  - LoadPassengersAction has the effect that a taxi agent will have loaded passengers after it has been performed.
  - LoadPassengersAction has the precondition that the bay it wishes to load at is accessible.
  - LoadPassengersAction has the precondition that the taxi must have allowed all passengers that wish to alight to do so before allowing new passengers to board the taxi.

- private void Update() - This method is called sixty times per second and checks whether the taxi is full or not. If the taxi is full, it alters the worldState of the Taxi to reflect that the taxi is full.

- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform LoadPassengersAction. It also asks the taxi agent which Bay would be an appropriate one for the taxi to perform LoadPassengersAction at. The position of the Bay is set as the target where the action is to be performed. This method is inherited from GoapAction.

- public override bool isDone() - This method returns a boolean that represents whether the LoadPassengersAction has been completed or not. It is inherited from GoapAction.

- public override bool perform(GameObject agent) - This method is executed continually once the taxi agent reaches an appropriate bay to load passengers at. Once the taxi becomes full, it sets the passengersLoaded attribute to true and alters the worldState of the taxi agent to reflect the relevant change that ocurred in the simulation world. It is inherited from GoapAction. If the taxi fails to perform the action, this method returns false. It otherwise returns true.

- public override bool requiresInRange() - This method defines whether the GoapAction LoadPassengersAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For LoadPassengersAction, requiresInRange() always returns true. It is a method inherited from GoapAction.

- public override void reset() - This method is used to reset the passengersLoaded attribute and the target of LoadPassengersAction. It is inherited from GoapAction.

A UML class diagram of class LoadPassengersAction can be seen in figure 4.13.

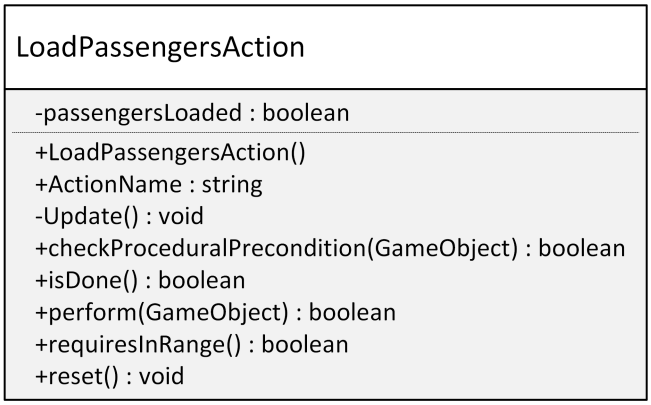


Figure 4.13: Class LoadPassengersAction

TaxiLeaveRankAction

Class TaxiLeaveRankAction is implemented as a subclass of class GoapAction. It represents the action of a taxi agent leaving the taxi rank via one of its exits. TaxiLeaveRankAction implements the following methods:

- public override string ActionName - This is a getter method and returns the string "TaxiLeaveRankAction". It is inherited from GoapAction.
- public TaxiLeaveRankAction() - This method is the constructor for a TaxiLeaveRankAction object. In it, the following preconditions and effects are described:
  - TaxiLeaveRankAction has the effect that a taxi agent will have left the rank with a full complement of passengers.
  - TaxiLeaveRankAction has the precondition that the taxi must have loaded a full complement of passengers.
- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform TaxiLeaveRankAction. It also determines where the taxi should exit the rank by setting an appropriate ExitNode as the target. This method is inherited from GoapAction.
- public override bool isDone() - This method returns a boolean that represents whether the TaxiLeaveRankAction has been completed or not. It is inherited from GoapAction.
- public override bool perform(GameObject agent) - This method is called when the taxi exits the simulation world. Since the recording of data regarding the taxi is performed by the Exit class which destroys the agent

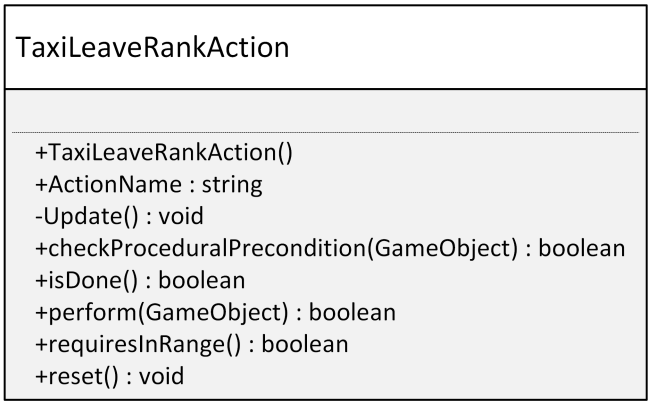


Figure 4.14: Class TaxiLeaveRankAction

once it reaches the exit, the contents of the perform method is left mostly empty. This is because a destroyed agent will not be able to perform in any meaningful way. It is inherited from GoapAction.

- public override bool requiresInRange() - This method defines whether the GoapAction TaxiLeaveRankAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For TaxiLeaveRankAction, requiresInRange() always returns true. It is a method inherited from GoapAction.
- public override void reset() - TaxiLeaveRankAction has no variables that need to be reset, so the body of this method is left blank. It is inherited from GoapAction.

A UML class diagram of class TaxiLeaveRankAction can be seen in figure 4.14.

AlightPassengersAction

Class AlightPassengersAction is implemented as a subclass of class GoapAction. It represents the action of allowing passengers to alight from the taxi. AlightPassengersAction implements the following methods:

- public override string ActionName - This is a getter method and returns the string "AlightPassengersAction". It is inherited from GoapAction.
- public AlightPassengersAction() - This method is the constructor for a AlightPassengersAction object. In it, the following preconditions and effects are described:
  - AlightPassengersAction has the effect that a taxi agent will have allowed all its alighting passengers to alight.

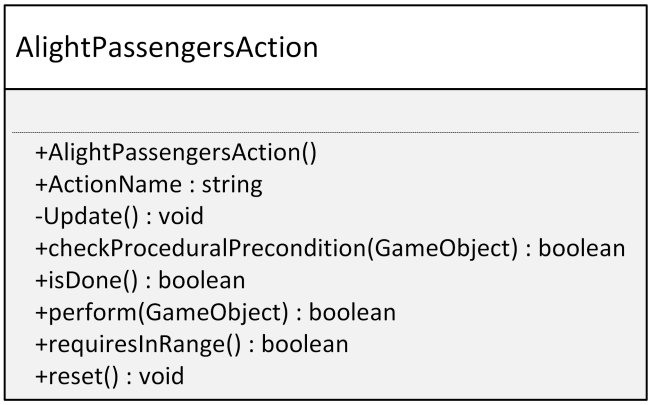


Figure 4.15: Class AlightPassengersAction

- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform AlightPassengersAction. It also determines where the taxi should stop to allow its passengers to alight. Usually, the taxi will stop at a bay where it plans to load new passengers. If a bay is not available, the taxi will seek alternate parking to stop and allow its passengers to alight.

- public override bool isDone() - This method returns a boolean that represents whether the AlightPassengersAction has been completed or not. It is inherited from GoapAction.

- public override bool perform(GameObject agent) - This method is executed when the taxi reaches a place where it can allow passengers to alight. The action sets the alightingPassengers attribute of the taxi agent to true. When the passengers have alighted, this method sets the passengers alighted variable to true to signal that the GoapAction is complete and the next action in the GoapAction sequence can be started.

- public override bool requiresInRange() - This method defines whether the GoapAction AlightPassengersAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For AlightPassengersAction, requiresInRange() always returns true. It is a method inherited from GoapAction.

- public override void reset() - AlightPassengersAction has no variables that need to be reset, so the body of this method is left blank. It is inherited from GoapAction.

A UML class diagram of class AlightPassengersAction can be seen in figure 4.15.

ParkTaxiAction

Class ParkTaxiAction is implemented as a subclass of class GoapAction. It is incorporated into the plan of a taxi agent when the GOAP planner deems it necessary for the taxi agent to wait for a bay to become accessible. ParkTaxiAction implements the following methods:

- public override string ActionName - This is a getter method and returns the string "ParkTaxiAction". It is inherited from GoapAction.
- public ParkTaxiAction() - This method is the constructor for a ParkTaxiAction object. In it, the following preconditions and effects are described:
  - ParkTaxiAction has the indirect effect that a Bay will become accessible.
- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform ParkTaxiAction. It then also sets an applicable parking space for the taxi agent as its target position.
- public override bool isDone() - This method returns a boolean that represents whether the ParkTaxiAction has been completed or not. It is inherited from GoapAction.
- public override bool perform(GameObject agent) - This method is executed when the taxi reaches the target parking space. It signals to the CarAIControl of the taxi agent to stop driving. It does so until it detects that a loading bay has opened. When it detects that a loading bay has become open, it signals to the CarAIControl that it can start driving the car again.
- public override bool requiresInRange() - This method defines whether the GoapAction ParkTaxiAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For ParkTaxiAction, requiresInRange() always returns true. It is a method inherited from GoapAction.
- public override void reset() - ParkTaxiAction has no variables that need to be reset, so the body of this method is left blank. It is inherited from GoapAction.

A UML class diagram of class ParkTaxiAction can be seen in figure 4.16.

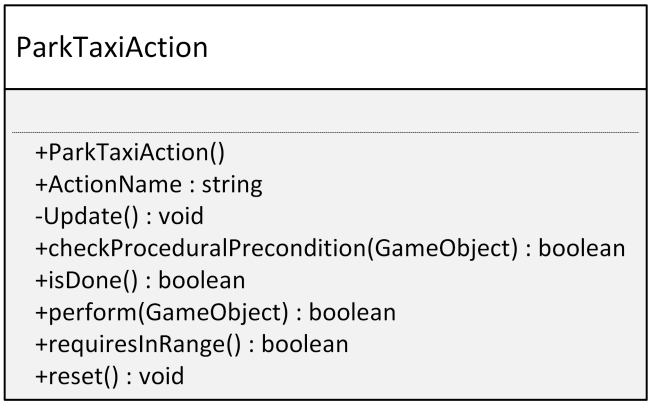


Figure 4.16: Class ParkTaxiAction

Commuter

Class Commuter is a model of a commuter at a taxi rank. It implements the interface IGoap. It has the following important attributes:

- private Destination destination - The destination to which the commuter desires to travel to.
- public bool inQueue - This boolean represents whether the Commuter is in a CommuterQueue or not.
- public Bay toEmbarkAt - This attribute is the Bay at which the commuter agent wants to get on to a taxi.
- private Dictionary<string, object> worldState - This attribute stores a representation of information regarding the simulation world that is relevant to the decision making process of the agent. It stores the relevant aspects of the simulation world as the commuter agent perceives them.
- private Dictionary<string, object> goalState - This attribute stores a representation of the current goals of the commuter agent.

Class Commuter also has the following important methods:

- public Bay getAppropriateBay() - This method searches for and returns a Bay where taxis going to the same destination that the commuter desires to travel to are going to load passengers.
- private void Start() - This method is called whenever a Commuter agent is instantiated. When called, it calls the initializeWorldState() and initializeGoalState() methods described below.

- private void initializeWorldState() - This method creates a new dictionary for the worldState attribute and adds the representation of what the agent perceives to be the world state to the dictionary. For a commuter arriving at the rank, the following world state is created:
  - It is false that the commuter has gotten to their destination.
  - It is false that the commuter has left the rank.
- private void initializeGoalState() - Here, the goals of the commuter are initialised. For the commuter agents in ACTS, the goal of a commuter is to get to their destination.
- public Dictionary<string, object> getWorldState() - This getter method provides external access to the worldState attribute. The use of the IGoap interface requires that this method be implemented.
- public Dictionary<string, object> getGoalState() - This getter method provides external access to the goalState attribute. The use of the IGoap interface requires that this method be implemented.
- public void planFailed(Dictionary<string, object> failedGoal) - The use of the IGoap interface requires that this method be implemented. In ACTS, if commuters fail to find a bay where they can be loaded to go to their desired destination, their goalState is changed. It is changed from wanting to leave the rank by taxi to wanting to leave the rank by foot.
- public void planFound(Dictionary<string, object> goal, Queue<GoapAction> actions) - The IGoap interface requires that this method be implemented. In ACTS, it is used by commuter agents to log the action steps that they plan to take to achieve their goals.
- public void actionsFinished() - The IGoap interface requires that this method be implemented. It is called when all the steps in the plan of an agent have been carried out by that agent. In ACTS, this method is used to log when an agent achieves its goals.
- public void planAborted() - The IGoap interface requires that this method be implemented. It is called by the GoapAgent when an action cannot be completed and the current plan must be abandoned. In ACTS it is used to log when such an event happens for an agent.
- public bool moveAgent(GoapAction nextAction) - This crucial method defines how an agent should move to the position where it can conduct the next action in its action sequence. The method first checks whether the target attribute of the AICharacterControl component attached to the agent is the same as the target of the nextAction. If it is not, the target

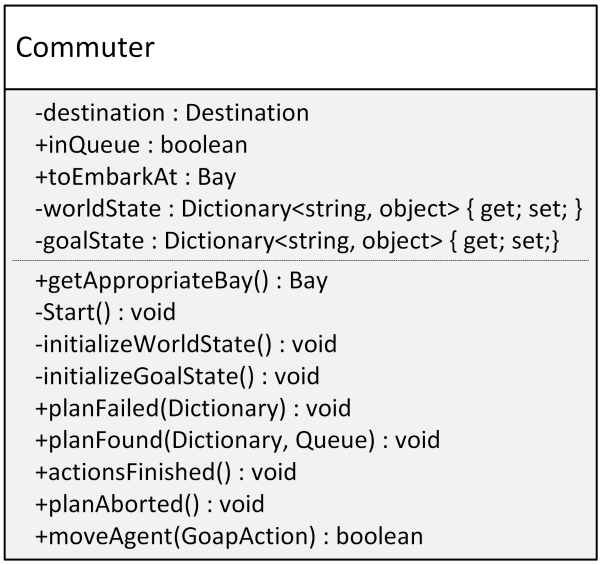


Figure 4.17: Class Commuter

of the AICharacterControl is changed to be the target of the nextAction. The moveAgent method then checks whether the commuter agent has arrived at the target of the nextAction or not. If it has arrived, the inRange attribute of nextAction is set to true and moveAgent returns true. If not, moveAgent returns false. The IGoap interface requires that this method be implemented.

A UML class diagram of class Commuter can be seen in figure 4.17.

GetOnAppropriateTaxiAction

Class GetOnAppropriateTaxiAction is implemented as a subclass of class GoapAction. It is incorporated into the plan of a commuter agent when the GOAP planner deems it necessary for the commuter agent to get on a taxi to satisfy the goals of the agent. GetOnAppropriateTaxiAction implements the following methods:

- public override string ActionName - This is a getter method and returns the string "GetOnAppropriateTaxiAction". It is inherited from GoapAction.
- public GetOnAppropriateTaxiAction() - This method is the constructor for a GetOnAppropriateTaxiAction object. In it, the following preconditions and effects are described:
  - GetOnAppropriateTaxiAction has the indirect effect that a Commuter gets to their destination.

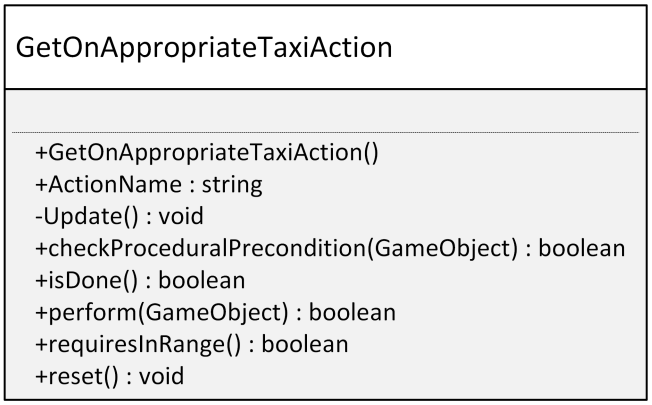


Figure 4.18: Class GetOnAppropriateTaxiAction

- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform GetOnAppropriateTaxiAction. It returns true if a Bay where taxis going to the same destination as the commuter exist at the taxi rank and false otherwise.
- public override bool isDone() - This method returns a boolean that represents whether the GetOnAppropriateTaxiAction has been completed or not. It is inherited from GoapAction.
- public override bool perform(GameObject agent) - This method is executed when the commuter reaches the target Bay. The commuter is then added to the CommuterQueue at that bay.
- public override bool requiresInRange() - This method defines whether the GoapAction GetOnAppropriateTaxiAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For GetOnAppropriateTaxiAction, requiresInRange() always returns true. It is a method inherited from GoapAction.
- public override void reset() - GetOnAppropriateTaxiAction has no variables that need to be reset, so the body of this method is left blank. It is inherited from GoapAction.
- private void Update() - This method is called sixty times per second. If the commuter is not already in the appropriate CommuterQueue, the action checks whether the position of the back of the CommuterQueue has changed. It updates its target to reflect the end of the CommuterQueue.

A UML class diagram of class GetOnAppropriateTaxiAction can be seen in figure 4.18.

CommuterLeaveRankAction

Class CommuterLeaveRankAction is implemented as a subclass of class GoapAction. It is incorporated into the plan of a commuter agent when the GOAP planner deems it necessary for the commuter agent to leave the rank by foot to satisfy the goals of the agent. CommuterLeaveRankAction implements the following methods:

- public override string ActionName - This is a getter method and returns the string "CommuterLeaveRankAction". It is inherited from GoapAction.
- public CommuterLeaveRankAction() - This method is the constructor for a CommuterLeaveRankAction object. In it, the following preconditions and effects are described:
  - CommuterLeaveRankAction has effect that the commuter leaves the rank.
- public override bool checkProceduralPrecondition(GameObject agent) - This method checks whether it is possible for an agent to perform CommuterLeaveRankAction. It also finds an appropriate place where the commuter can exit the taxi rank.
- public override bool isDone() - This method returns a boolean that represents whether the CommuterLeaveRankAction has been completed or not. It is inherited from GoapAction.
- public override bool perform(GameObject agent) - This method is executed when the commuter reaches the target exit. The body of this method is left empty as there is nothing for the commuter to do once the rank has been left.
- public override bool requiresInRange() - This method defines whether the GoapAction CommuterLeaveRankAction can be performed anywhere in the simulation world or whether it needs to be performed in range of a target position. For CommuterLeaveRankAction, requiresInRange() always returns true. It is a method inherited from GoapAction.
- public override void reset() - CommuterLeaveRankAction has no variables that need to be reset, so the body of this method is left blank. It is inherited from GoapAction.

A UML class diagram of class CommuterLeaveRankAction can be seen in figure 4.19.

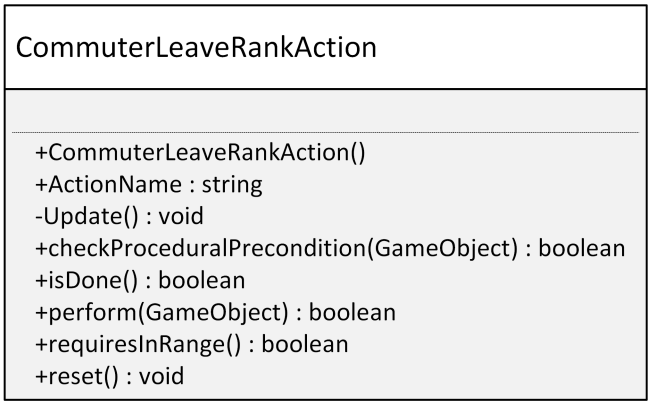


Figure 4.19: Class CommuterLeaveRankAction

IOACTS

Class IOACTS handles user input for the ACTS system. It allows a user to draw the Vector3Graph used by the taxi navigation system. It also makes provision to save the Vector3Graph drawn. In addition, any saved Vector3Graph can be loaded by IOACTS upon simulation start up. This eliminates the need to continually redefine the Vector3Graph. The loaded Vector3Graph is also then displayed visually with NavNode, BayNode, ParkingNode and ExitNode prefabs. These have the appearance of coloured spheres. The edges of the Vector3Graph are displayed as lines that link the spheres drawn. The lines transition from a blue colour to a grey colour over their length. The blue colour is used at the start of the edge and the grey colour at the end. This allows the user to tell in which direction the edge is. IOACTS has the following attributes:

- public static Vector3Graph carNavGraph - This attribute stores the Vector3Graph to be used during a simulation run.
- public static clickState state - This static attribute keeps track of what input state the simulation is in. The type "clickState" is an enum type consisting of:
  - AddNavNode - In this state, ACTS accepts input to create a Node for the carNavGraph attribute of IOACTS.
  - AddNavEdge - In this state, ACTS accepts input to create an Edge for the carNavGraph attribute of IOACTS.
- public GameObject NavNode - This attribute stores the prefab from which to clone all Node instances needed for the visual representation of the carNavGraph.

- public GameObject BayNode - This attribute stores the prefab from which to clone all BayNode instances needed for the visual representation of the carNavGraph.
- public GameObject ParkingNode - This attribute stores the prefab from which to clone all ParkingNode instances needed for the visual representation of the carNavGraph.
- public GameObject ExitNode - This attribute stores the prefab from which to clone all ExitNode instances needed for the visual representation of the carNavGraph.
- public int clickNumber - This integer attribute stores whether the user is busy defining the start of an Edge or the end of an Edge.

IOACTS has the following methods as well:

- private void Start() - This method is called during program start up. It loads a saved Vector3Graph into the carNavGraph attribute and then creates a visual representation of the graph.
- private void Update() - This method is called sixty times per second and detects whether the user is providing input or not. It then creates an appropriate Node or Edge as directed by the user and adds the created Node or Edge to carNavGraph. When an Edge is created, a copy of the updated carNavGraph is saved to disk.
- private void drawNavLine(Edge edge) - This method accepts an Edge as a argument and draws a visual representation of the Edge in the simulation world. The visual representation is a three dimensional line that transitions from blue to grey in colour. The blue side of the line represents the start of the Edge and the grey side represents the end of the Edge.

A UML class diagram of class IOACTS can be seen in figure 4.20.

ACTSScheduler

Class ACTSScheduler is responsible for scheduling when what events occur during a simulation run of ACTS. Examples of events include the arrival of commuters and the arrival of taxis in the taxi rank. ACTSScheduler defines the following attributes:

- public GameObject exampleTaxi - This attribute is used to store the prefab from which all taxi agents are to be cloned and instantiated from.
- public GameObject exampleCommuter - This attribute is used to store the prefab from which all commuter agents are to be cloned and instantiated from.

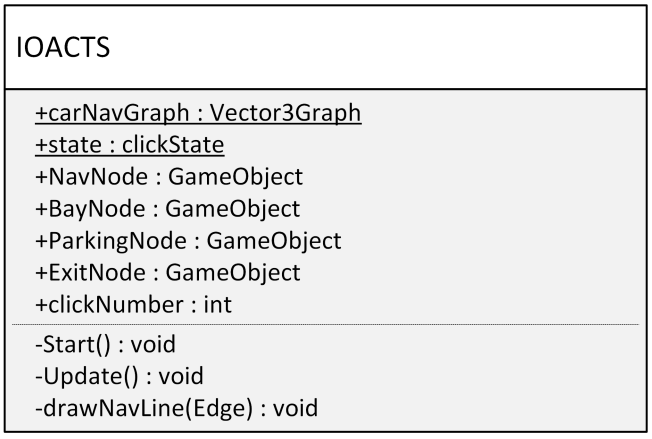


Figure 4.20: Class IOACTS

- public static float firstArrivalTime - This attribute stores the time that the first taxi will arrive at the rank at.

ACTSScheduler also has the following methods:

- private IEnumerator TaxiSpawn(GameObject taxi, float time) - This method is used to schedule when a taxi should arrive at the rank. The taxi given as its first argument and is scheduled to arrive at the time, in seconds, after the start of the simulation specified by its second argument.
- private -IEnumerator SpawnCommuter(Destination destination, float time) - This method is used to schedule when a commuter should arrive at the rank. The destination that the commuter should travel to is given as its first argument and the commuter is scheduled to arrive at the time, in seconds, after the start of the simulation specified by its second argument.
- private LinkedList<GameObject> SpawnInactivePeopleInTaxi(int number) - This method creates a list of commuter agents that a taxi can save as its list of passengers. It is usually used when a taxi agent is initialised and needs to be provided with passengers.
- private void Start() - This method is called on application start up. The first thing it does is to set the passage of time in the simulation to six times that of reality. This is to save time when conducting simulation runs. It also adjusts the friction coefficients of the tires of the example taxi so that they can deal realistically with the change in the speed of time. This method then loads the observed data discussed in section 4.2 into memory. Each record of observed data is used to schedule when a

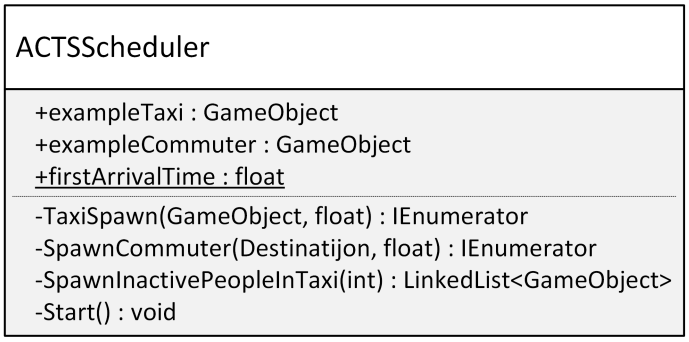


Figure 4.21: Class ACTSScheduler

taxi should arrive at the rank. The TaxiSpawn method is used to do this. The fields of the record are used to provide initial data for the attributes of the taxi agent that is to arrive at the rank. The data concerned consists of:

- The time at which the taxi should arrive at the rank
- The number of passengers aboard the taxi
- The fare charged by the taxi
- The destination that the taxi will travel to
- The route number allocated to the taxi
- The maximum number of passengers that the taxi can seat
- An identifying string for the taxi
- The time at which the taxi was observed departing

Finally, this method schedules the arrival times of all commuter agents that will arrive at the rank. For more details regarding this process, see section 4.7.

A UML class diagram of class ACTSScheduler can be seen in figure 4.21.

4.7 Important algorithms

This section describes the important algorithms developed for the functioning of ACTS.

Taxi navigation

The taxi navigation system was created so that taxi agents could effectively and realistically drive to their desired targets in the simulation world. A directed graph serves as the basis for the navigation system. In this graph, the nodes

represent target positions in the simulation world. The edges in the graph represent the straight line road segments that connect nodes. The edges as described in section 4.6 have a start and an end. The navigation system only allows paths that can be formed by travelling from the starts of edges to the ends of edges. The classes defined for the graph, nodes and edges are described in section 4.6.
The first step in the navigation process is to determine how far the taxi agent is from its desired position. If the distance is less than a defined threshold, it is not necessary to navigate any further and the navigation process can be stopped. If the distance to the target is greater than the threshold, the agent checks whether it is already navigating to the target. If that is the case, the navigation system realises that it does not have to recalculate a path to the target and allows the agent to continue driving along the path it already is following. If the navigation system realises that the taxi agent is not navigating to its desired target, it identifies which Node in the graph has the same position as where the agent would like to be in the simulation world. The agent then calls its "planRoute(Node endNode)" method with the Node it identified as the argument. This method finds a valid path through carNavGraph that will lead the taxi agent to its target. It then engages the CarAIControl component of the taxi agent to start providing appropriate accelerator, braking and steering inputs to lead the taxi agent along the identified path. A flow chart describing the taxi navigation algorithm can be seen in figure 4.22.

Loading commuters

The process for loading a commuter starts when a commuter agent decides to get on an appropriately identified taxi. The commuter agent identifies the queue of commuters in the simulation world that wish to board the identified taxi. The NavMeshAgent component of the commuter agent is then given the last commuter in the identified queue as a target. The target is updated whenever another commuter joins the queue. When the commuter agent reaches the back of the queue, it becomes the back of the queue. The commuter agent then waits in the queue until it reaches the front of the queue. While waiting in the queue, the NavMeshAgent component ensures that the commuter agent follows the commuter agent directly in front of it in the queue. When the commuter agent reaches the front of the queue, the addPassengerFromQueue(CommuterQueue queue) of the taxi agent waiting there is called. This method then removes the commuter agent from the queue and loads the commuter into its list of passengers by the following process:

1. The taxi agent checks whether there is still space to load a passenger

2. The taxi agent checks whether another commuter agent is not being loaded

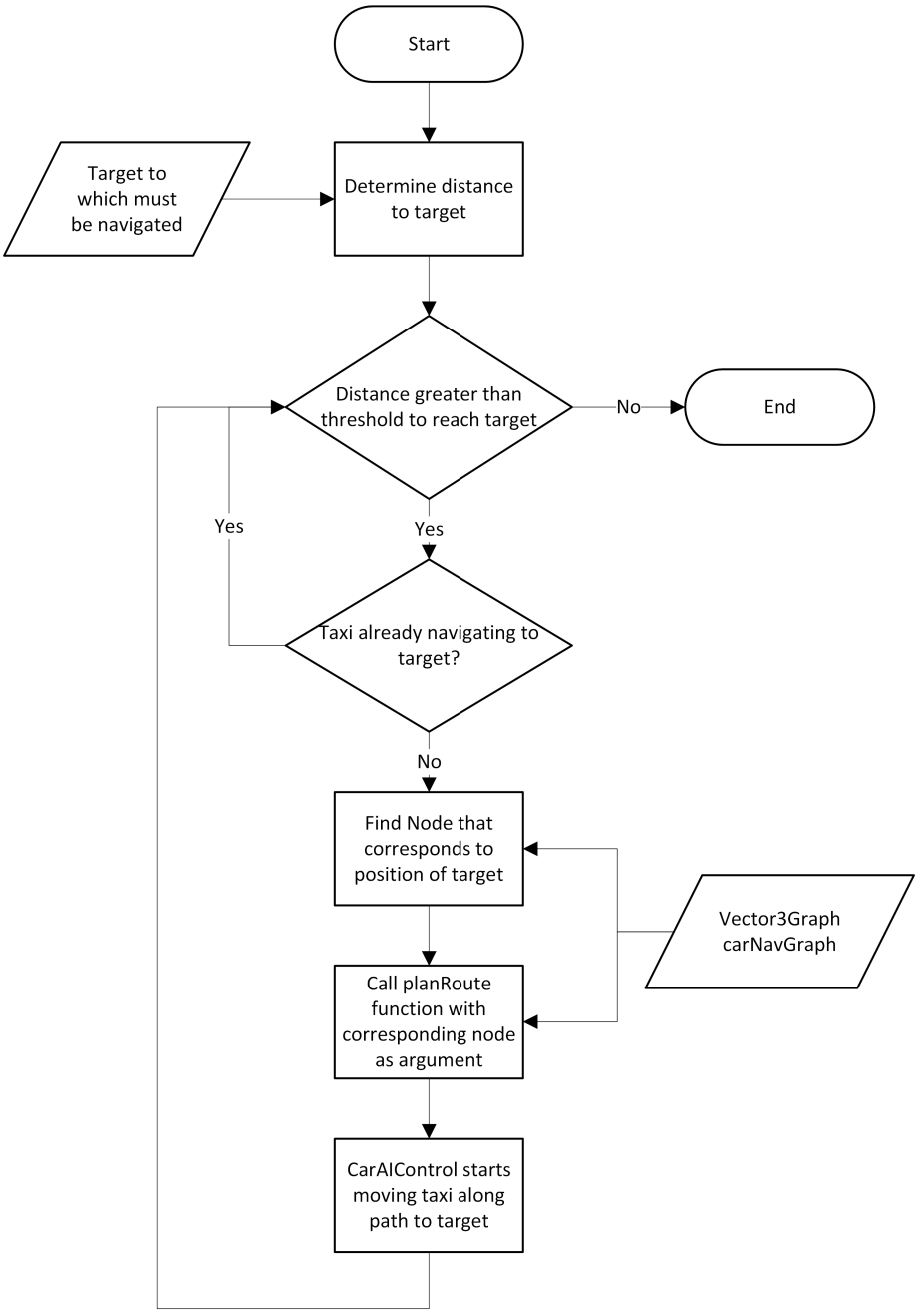


Figure 4.22: Taxi navigation algorithm

3. The time it will take to load the commuter is generated from an inverse log-normal distribution with a mean of 7.87 seconds and a standard deviation of 2.06 seconds. The use of this distribution was suggested by (van-Biljon and CJ Venter 2013).

4. Once the generated time has elapsed, the first commuter agent is removed from the CommuterQueue. It is then added to the list of passengers of the taxi and set to be inactive. An inactive GameObject is not visible and does not interact with the simulation world.

A flow chart describing the commuter loading algorithm can be seen in figure 4.23.

Taxi Parking

This algorithm defines the procedure a taxi agent follows when it determines that it needs to wait for a loading bay to become available. The first step is to navigate to the parking area at the taxi rank. The taxi agent then waits in the parking until it detects that a loading bay has become available. Once a loading bay becomes available, the taxi navigation system is used to move the taxi agent to the bay. A flow chart describing the algorithm that taxis use to wait for a loading bay to open can be seen in figure 4.24.

Data logging system

The data logging system maintains a comma separated value (CSV) file that contains the important outputs generated by ACTS. The logging system continually checks whether a taxi agent reaches an exit of the taxi rank. Once a taxi agent reaches an exit, the taxi agent is destroyed after the following information about that taxi agent has been saved to the output file:

- The time that the taxi agent exits the rank
- The time that the taxi agent had entered the rank
- A unique ID associated with the taxi agent
- The destination to which the taxi agent will travel
- The departure time expected for the taxi agent as observed and recorded in the taxi agent rank survey used as input for the simulation.

A flow chart describing the process by which data regarding taxi agents is logged can be seen in figure 4.25.

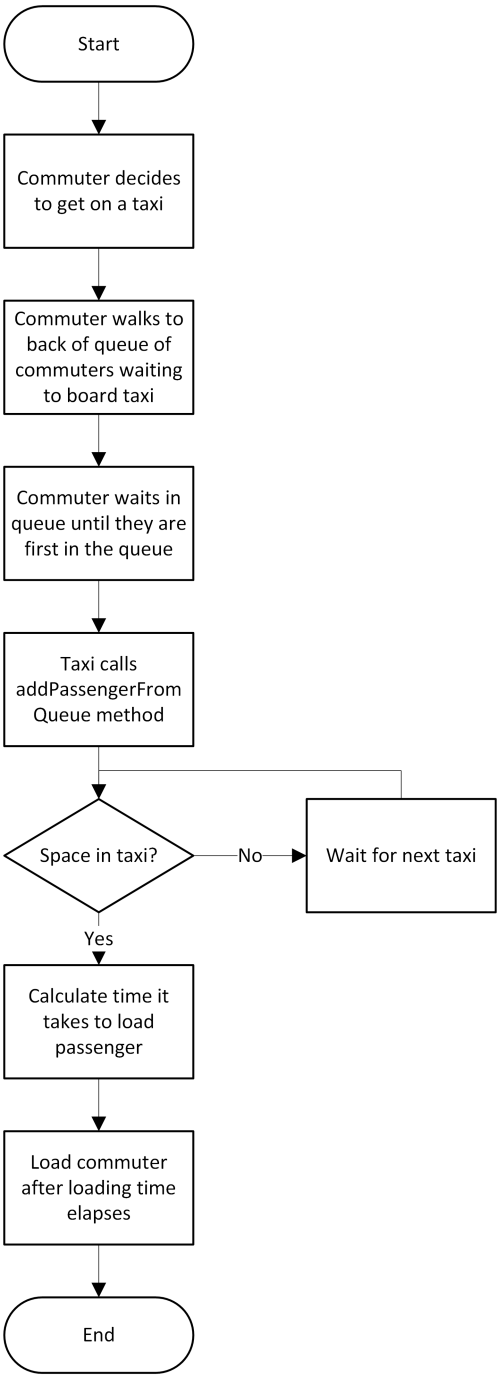


Figure 4.23: Commuter loading algorithm

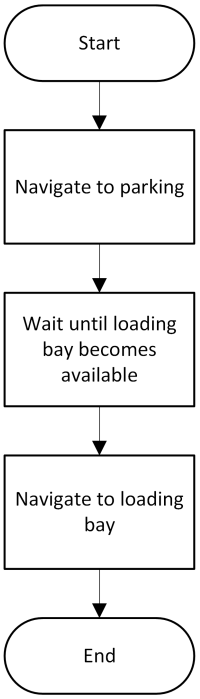


Figure 4.24: Taxi waiting algorithm

Taxi arrival scheduling

The scheduling of taxis arriving at the rank is performed upon application start up in class ACTSScheduler. The first step is to load a taxi rank survey into memory. The taxi rank survey loaded for this study is the survey performed for Stellenbosch Municipality as described in section 4.2. The records of the survey represent taxis and are used to instantiate taxi agents. The field that describes the time that a taxi joins the queue is used to determine when the taxi should be instantiated as an agent in ACTS. The TaxiSpawn(GameObject taxi, float time) method of class ACTSScheduler is called to ensure that the taxi agent appears at the rank at the appropriate time. A flow chart describing the process by which taxi agents are scheduled to arrive can be seen in figure 4.26.

Commuter arrival scheduling

The main assumption made regarding the arrival of commuters is that their inter arrival times are random. This assumption is recommended by (van-Biljon and CJ Venter 2013). They suggest calculating inter arrival times by the following procedure:

1. Determine the mean rate of arrivals per second, Q, for the study period

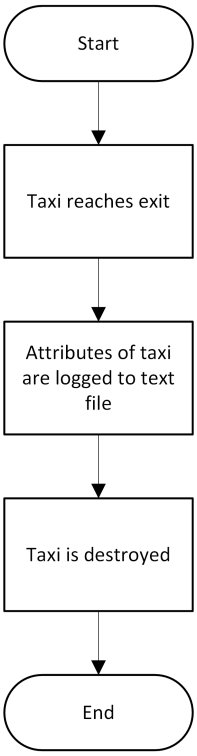


Figure 4.25: Taxi data logging system

2. Calculate an inter arrival time by using an exponential distribution: $\Delta t_i = (-1/Q) * \ln(1 - R_i)$ where R_i is a random real number between 0 and 1

3. Repeat until all inter arrival times have been calculated

For ACTS, this process was modified in the following ways:

- The commuter flux, Q, was determined per destination.

- The flux was recalculated each time an inter arrival time was calculated based on the number of commuters expected in the fifteen minute period before the time of arrival of that commuter; t.

The number of taxis going to each destination was scaled by fifteen to approximate the number of commuters expected to be travelling to each destination. This was done as each taxi can legally seat fifteen passengers. A flow chart describing the process by which commuter agents are scheduled to arrive can be seen in figure 4.27.

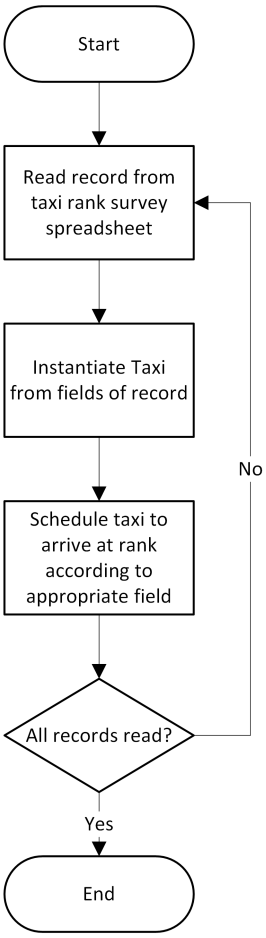


Figure 4.26: Taxi arrival scheduling algorithm

Taxi sensing

To ensure that they do not collide with other taxi agents in the simulation, taxi agents need to sense what is ahead of them. This is done by projecting a straight line ahead of a taxi agent. The length of the line is linearly proportional to the speed of the taxi. The direction that the line points corresponds to the current steering angle of the taxi. If the line intersects with the Collider component of any other taxi agent GameObject, the distance between the agents as well as their current velocities are used as inputs to the car following model. The car following model then returns a desired acceleration that will allow the taxi agent to avoid colliding with the taxi agent ahead of it. The desired acceleration is used by the CarAIControl component of the taxi agent to provide the CarController component of the taxi agent with appropriate accelerator or braking input. The car following model implemented can be seen in equation

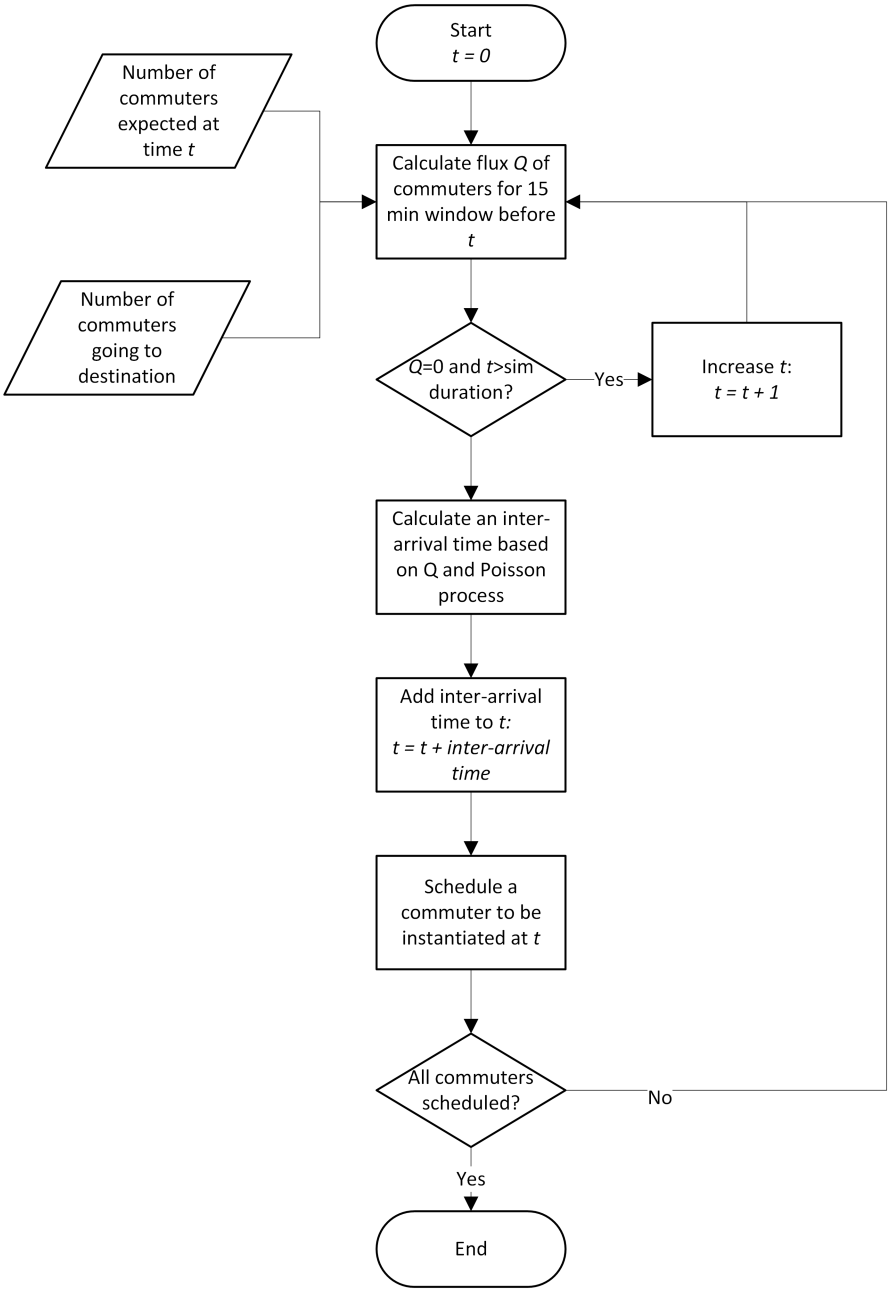


Figure 4.27: Commuter arrival scheduling algorithm

4.1:

$$a_{n+1}(t+T) = \frac{\lambda}{\|\vec{s}_n(t) - \vec{s}_{n+1}(t)\|}(\|\vec{v}_n(t)\| - \|\vec{v}_{n+1}(t)\|) \tag{4.1}$$

Where:

- t is the current time step of the simulation.
- T is the time step increment applied in the simulation.
- $a_{n+1}(t+T)$ is the desired acceleration or deceleration determined by the car following model for the taxi that detected another taxi ahead of it.
- λ is a variable calibrated to determine how sensitive the taxi agent is to a stimulus. It was calibrated to 0.7 in ACTS.
- $\vec{s}_n(t)$ refers to the cartesian position vector of the leading taxi agent at time t.
- $\vec{s}_{n+1}(t)$ refers to the cartesian position vector of the following taxi agent at time t.
- $\vec{v}_n(t)$ refers to the cartesian velocity vector of the leading taxi agent at time t.
- $\vec{v}_{n+1}(t)$ refers to the cartesian velocity vector of the following taxi agent at time t.

A flow chart describing the process by which taxi agents sense traffic ahead of them and react accordingly can be seen in figure 4.28.
The implementation of ACTS developed in this chapter is tested in the chapter that follows.

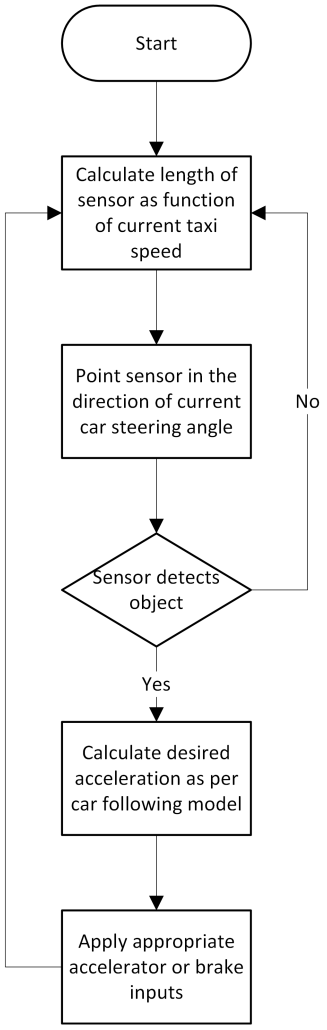


Figure 4.28: Taxi sensing algorithm

Chapter 5

Model validation

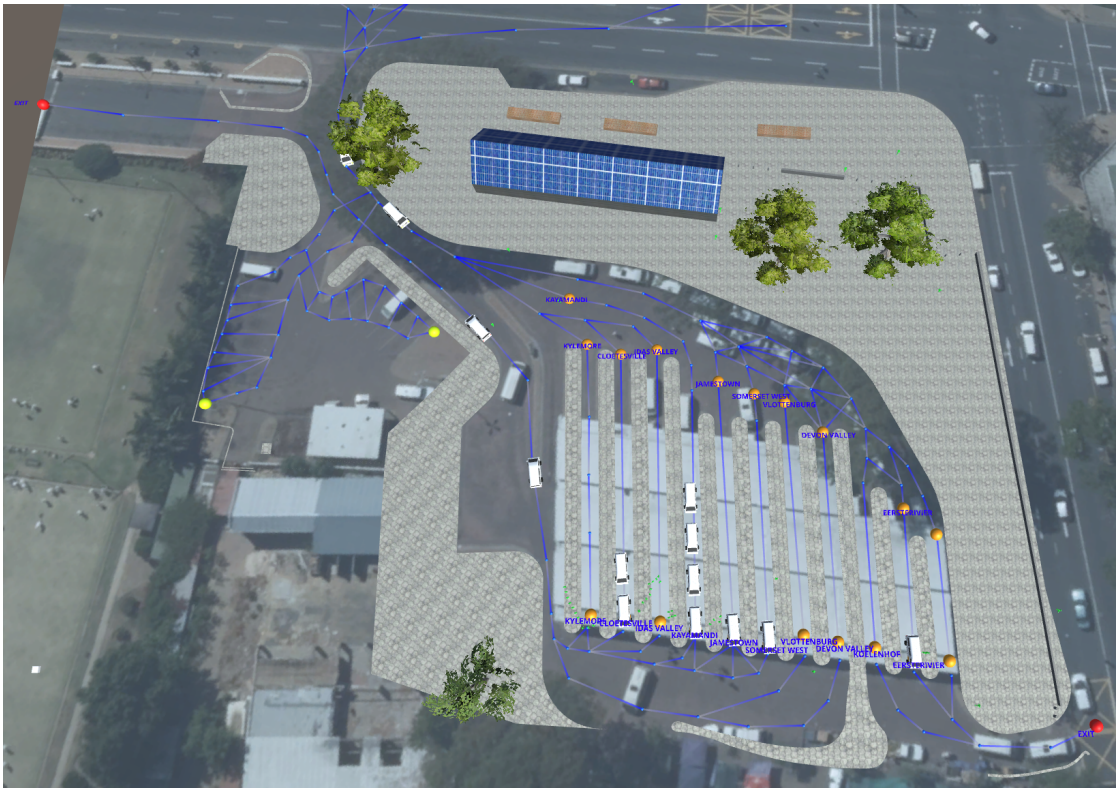


Figure 5.1: General appearance of ACTS visual output

This chapter is concerned with establishing the validity of ACTS. It starts with a discussion on the methods available to test simulation models. This is followed by a section detailing which techniques were found to be relevant for the testing of ACTS. Finally, a description of the results of each technique

applied during the validation of ACTS is provided. More than one hundred simulation runs of each peak period were conducted and the results of each were recorded for analysis and comparison.

5.1 Validation of simulation models

Simulation model validation is the process of testing the accuracy and credibility of a simulation model. (Sargent 2013) describes the following validation techniques for simulation models.

- Animation - Showing a visual representation of the model as it appears as time passes.
- Comparison to other models - The results of the model can be compared to previously validated models. This will provide an indication of the validity of the model.
- Degenerate tests - Providing the model with inputs that have an expected outcome. If the model does not produce conforming outputs, its behaviour is considered degenerate. An example is that we expect the lengths of queues to increase when the arrival rate is greater than the queue service rate. If this is not observed, the model has degenerate behaviour.
- Event validity - The events that occur in the model are compared to what one would expect in reality.
- Extreme condition tests - The behaviour and outputs of the model when provided with extreme inputs are considered.
- Face validity - Experts in the field being modelled are queried as to whether the assumptions used to form the model are valid. They can also be queried as to whether the output provided by the model is reasonable.
- Historical data validation - The outputs of the model for certain input conditions are compared with the corresponding historical records for the same input conditions.
- Historical methods:
  - Rationalism: If it is known that the assumptions that a model is built on are true, then by deduction a valid model was produced.
  - Empiricism: The assumptions and outputs are compared to empirical observations.
  - Positive economics: The validity of the assumptions is disregarded in favour of only requiring the outcomes to be valid.

- Multistage validation - Applying the historical methods as three stages:
  1. Use rationalism to develop the assumptions of the model
  2. Use empiricism to test the assumptions of the model
  3. Validate the outcomes of the system as per positive economics
- Internal validity - If several runs of a simulation for the same input conditions produce outcomes that vary greatly, the model lacks consistency. This reflects poorly on the credibility of the model if consistent results are expected.
- Graphical comparisons of data - Histograms, box and whisker plots and scatter plots of the model and the system being modelled can be created and compared.
- Operational graphics - Figures dynamically describing the state of the model as it passes through time can be assessed to determine the validity of the model.
- Sensitivity analysis - The various input parameters of the model can be changed and the effect of those changes on the results of the model can be established. Parameters that have strong effects on the results produced by the simulation model should be scrutinised and ensured to be accurate.
- Predictive validation - The model and the system being modelled can be provided with the same inputs. The forecast of the model and the behaviour of the system can then be compared.
- Traces - The behaviour and state of the entities in the simulation model can be recorded throughout the simulation model. This "trace" data can then be analysed to determine whether the model is behaving sensibly.
- Turing tests - Experts in the field being modelled are asked whether they can discriminate between the outcomes of the system and the outputs of the model.

5.2 Relevant validation techniques

For the validation of ACTS, the following validation techniques were chosen:

- Animation - This technique was chosen as it readily reveals whether the algorithms in ACTS operate as intended. It also provides the ability to see that the simulation world is defined by the rank geometry as was expected. The use of animation to validate ACTS is described in section 5.3.
- Graphical comparisons of data - This technique was chosen as data concerning the operation of Bergzicht taxi rank was available for the AM peak time, PM peak time and Saturday peak time. Figures created from this data could be compared to figures created from the corresponding outputs produced by ACTS. The use of this technique to validate ACTS is described in section 5.4.
- Historical data validation - This technique was chosen as historical records concerning the operation of Bergzicht taxi rank for an AM peak time, a PM peak time and a Saturday peak time were available. The average time that a taxi spent in a rank was determined from the historical data and compared to the average time predicted by the model for each peak time scenario. The comparisons were done by means of statistical hypothesis testing as well as confidence interval testing. Section 5.5 details the comparisons that were made.

5.3 Interpretation of visual output

General comments

The appearance of the visual output provided by ACTS can be seen in figure 5.2.

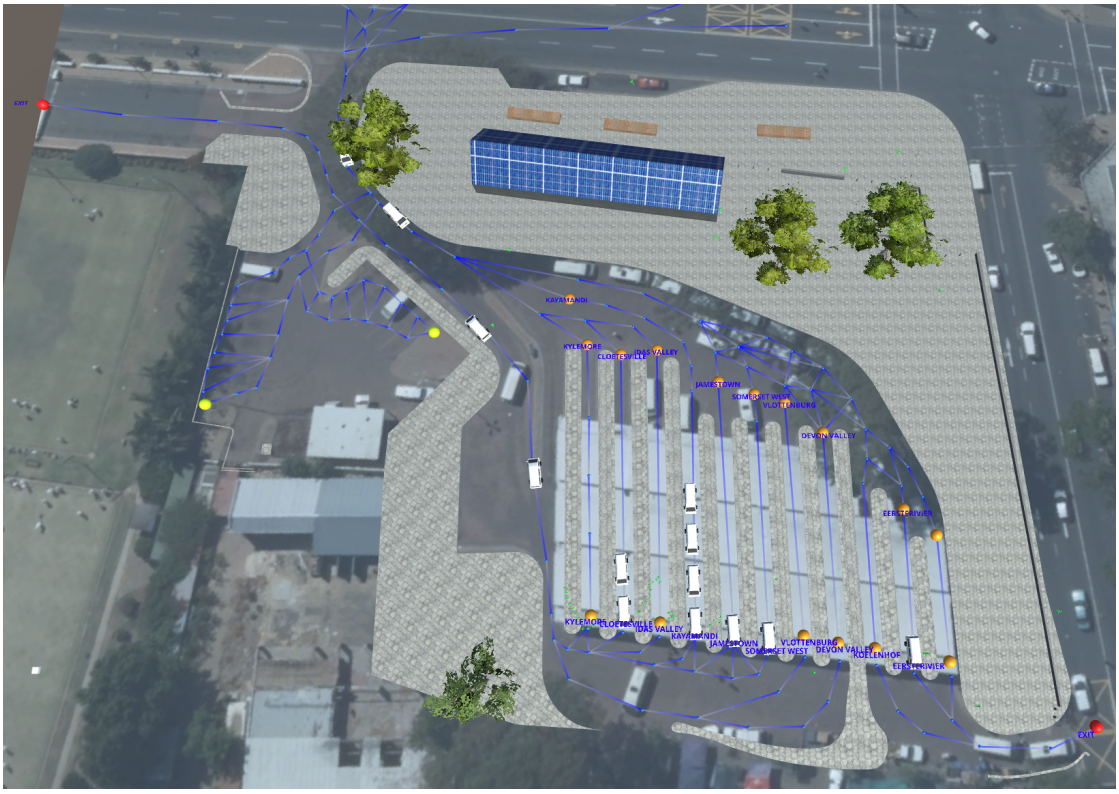


Figure 5.2: General appearance of ACTS visual output

The base plane of the simulation is overlaid with a satellite image that corresponds to the Bergzicht taxi rank area. The paving of the rank is rendered with a grey concrete block texture. The walls and bollards of the rank are rendered with a concrete texture. The roof of the building located on the rank premises is rendered with a solar panel texture. The choice of these textures is arbitrarily aesthetic and have no effect on the functioning of ACTS. They do however, aid in discerning the various features modelled. The appearance of the paving, walls and three bollards can be seen in more detail in figure 5.3.

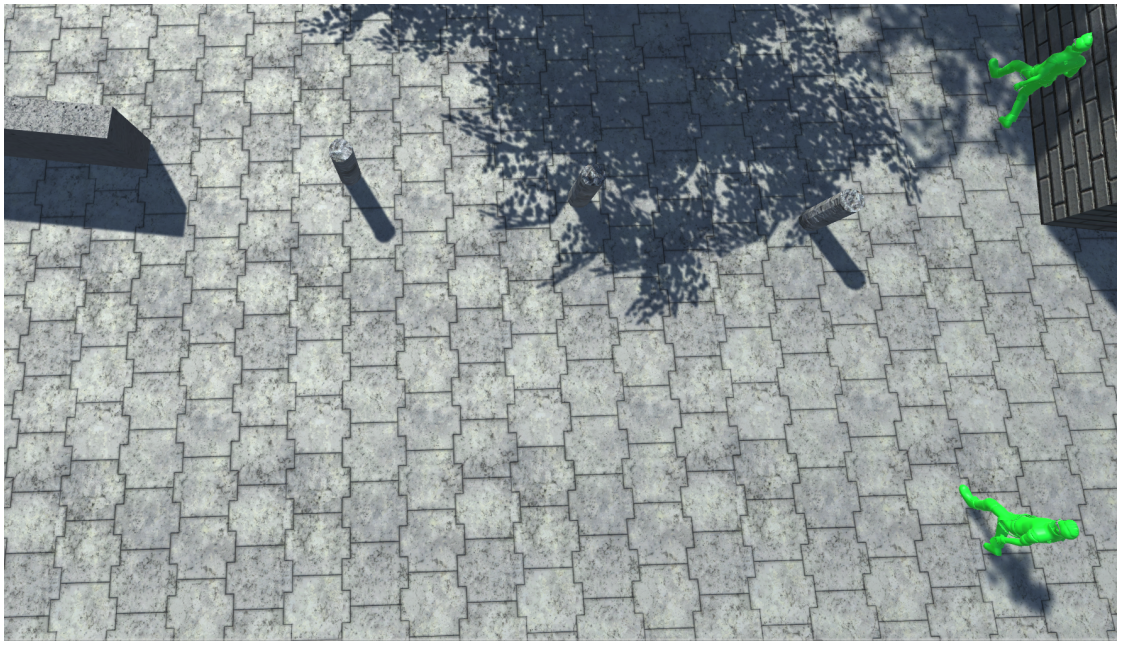


Figure 5.3: Taxi rank paving and bollards

Trees as provided by the Unity Standards Assets package were also placed according to the positions of real trees at the rank. The three dimensional nature of the paving, walls, bollards and building is made apparent by the shadows that they cast in the simulation world. Figure 5.4 shows how the three dimensional geometry was used to determine where commuters could and could not form paths to walk. Blue areas represent areas where the commuters can walk and the other areas are where it was determined that commuters cannot walk.

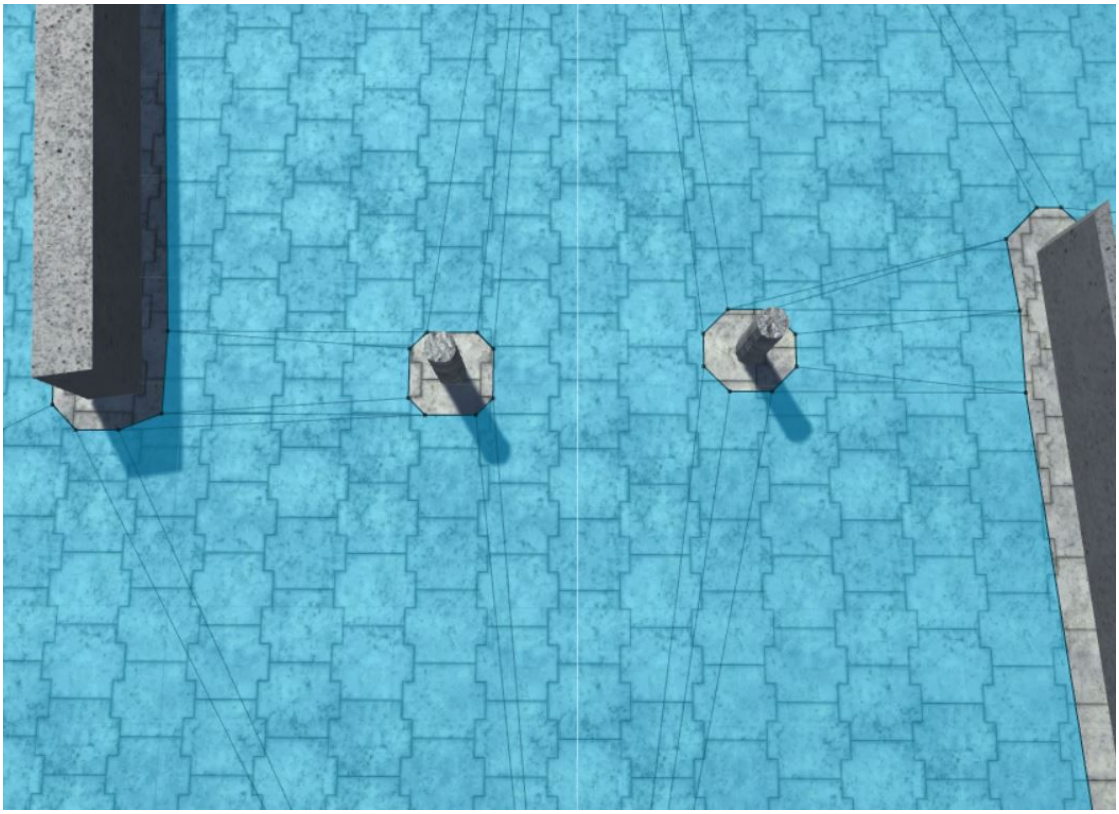


Figure 5.4: NavMesh around bollards and walls

Taxi navigation

The nodes and edges that carNavGraph consists of are of critical importance to the ability of taxi agents to navigate in ACTS. Nodes were visually represented by coloured spheres. Blue spheres represent instances of class Node, orange spheres represent instances of class BayNode, yellow spheres represent instances of class ParkingNode and red spheres represent instances of class ExitNode. Edges were visually represented by lines that connect the nodes in carNavGraph. A colour gradient that shifts from blue to grey across a line is used to discriminate between the start and end of an Edge. The blue side of a line represents the start of an Edge and the grey side of a line represents the end of an Edge. Figure 5.5 shows the visual output representing nodes and edges.

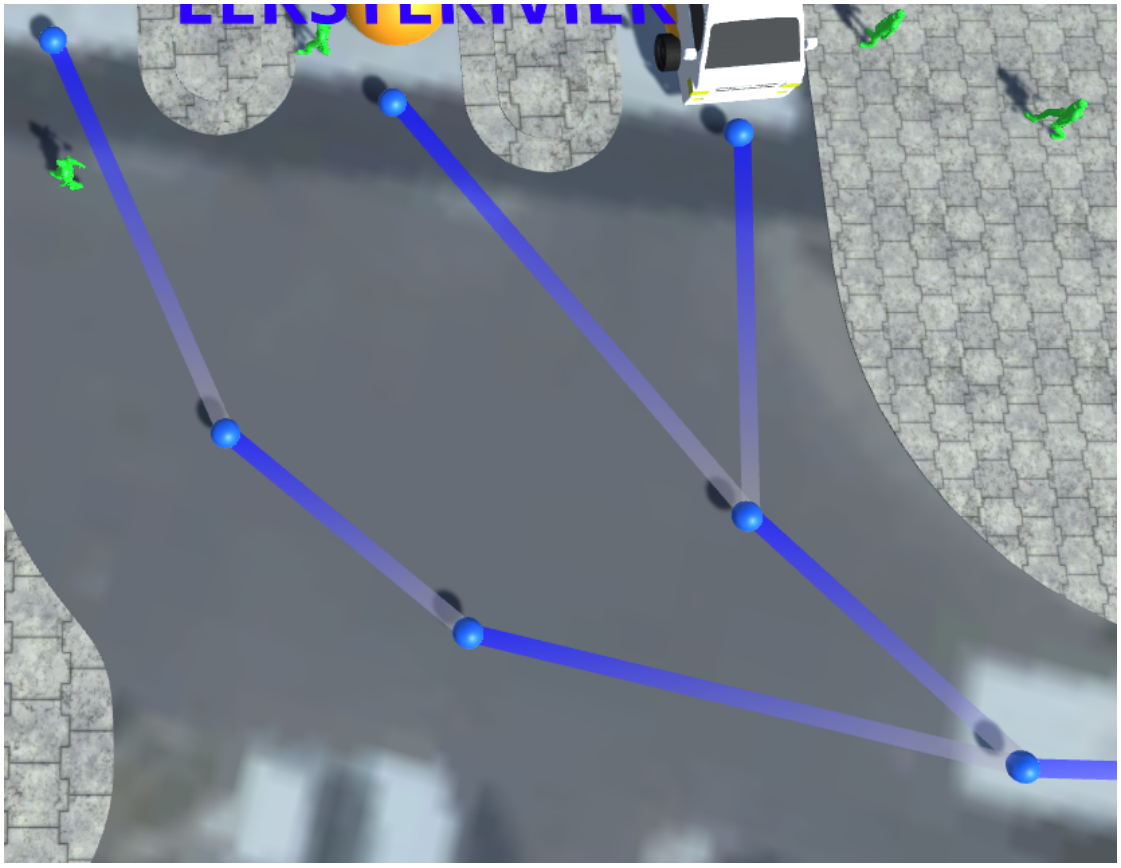


Figure 5.5: Visual representation of Node and Edge classes

Figure 5.6 shows the visual representation of the BayNode class. In addition to the orange sphere that represents the position of the BayNode, blue text rendered below the sphere indicates which destination the BayNode is associated with.

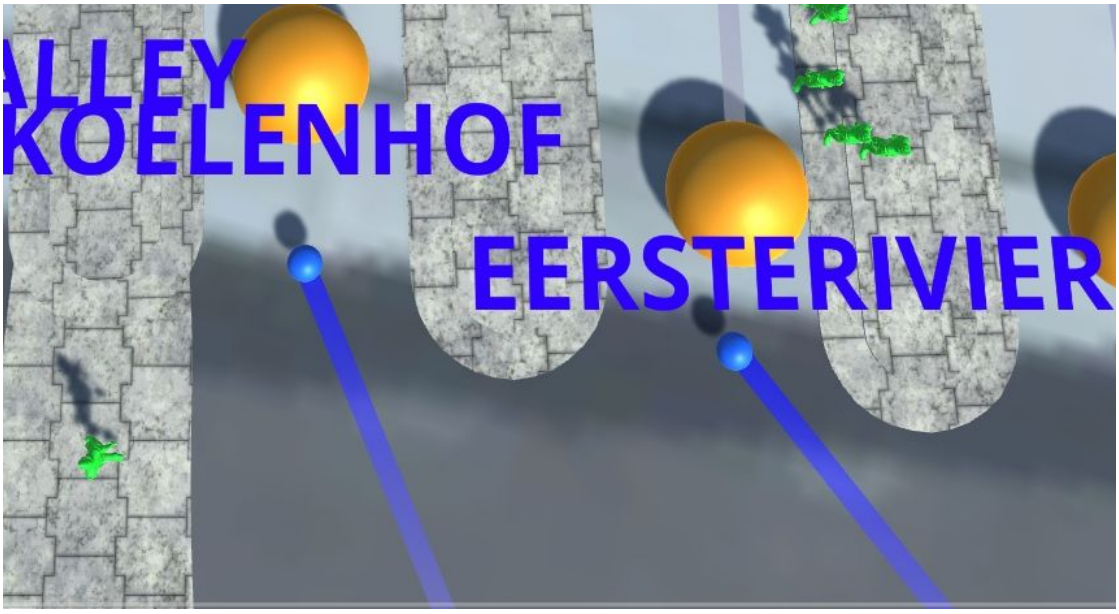


Figure 5.6: Visual representation of BayNode class

Figure 5.7 shows the visual representation of a ParkingNode. It was seen to be positioned as intended.

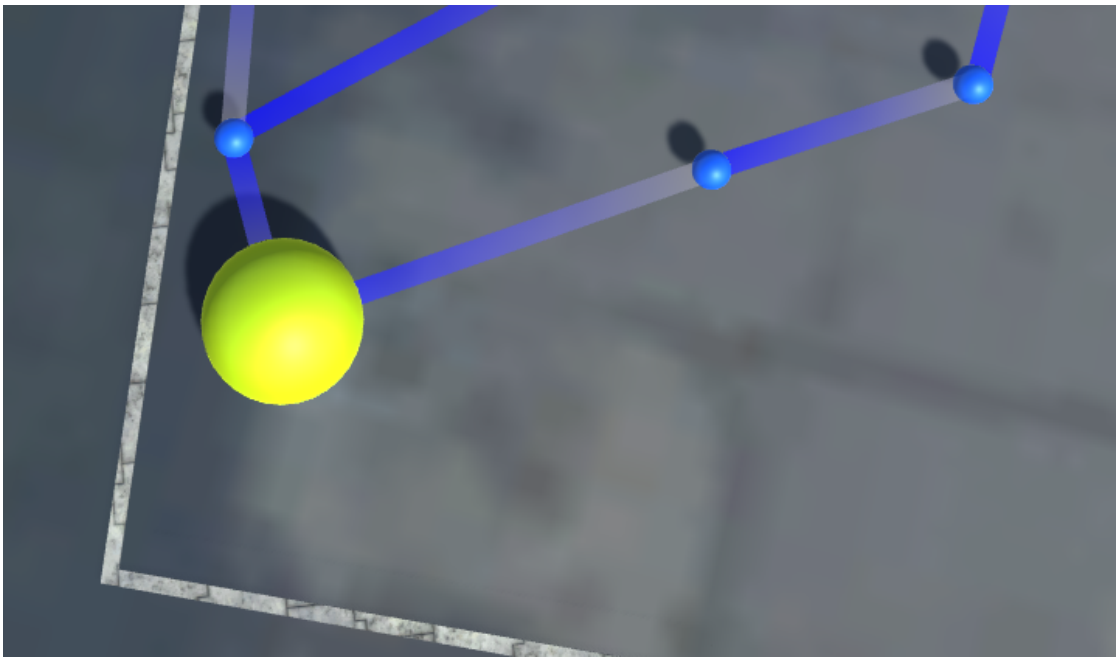


Figure 5.7: Visual representation of ParkingNode class

The visual representation of class ExitNode can be seen in figure 5.8. Blue text to the left of the visual representation also indicates that this is an exit of the taxi rank.

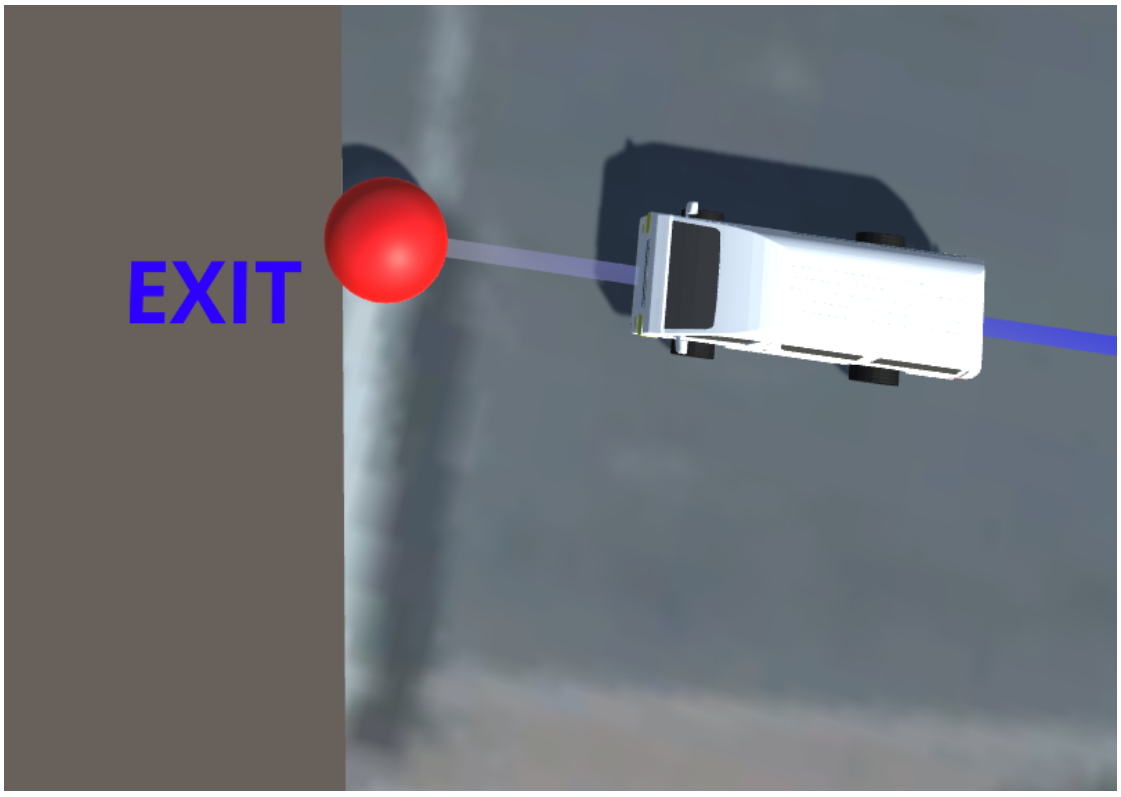


Figure 5.8: Visual representation of ExitNode class

The positioning of the various nodes described above as well as how they are connected as shown by the lines were inspected and found to be satisfactory. Taxi agents were then allowed into ACTS. Figure 5.9 shows that the taxi agents navigated the rank as was desired. They followed the shortest paths in carNavGraph that led to the positions that they wanted to navigate to, i.e., the various BayNodes.

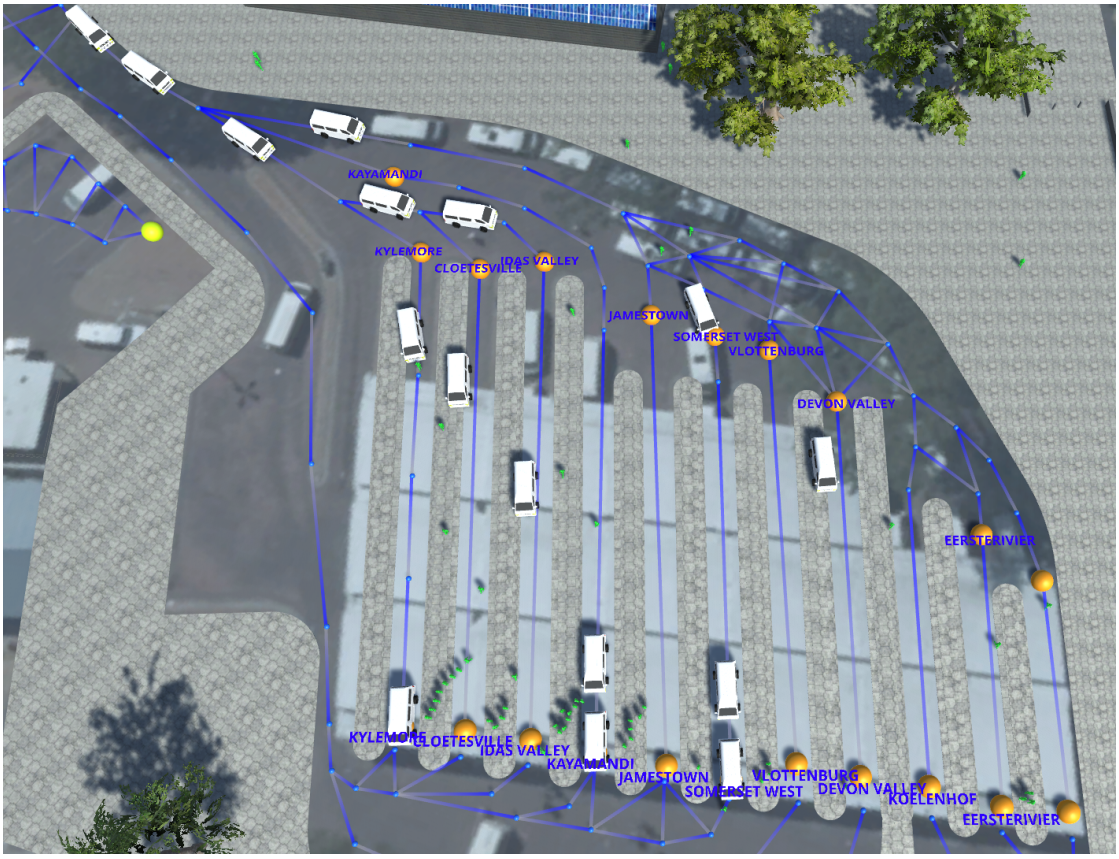


Figure 5.9: Visual output showing taxis navigating

Figure 5.10 shows that the taxi agents are also able to navigate to an appropriate exit after loading passengers.

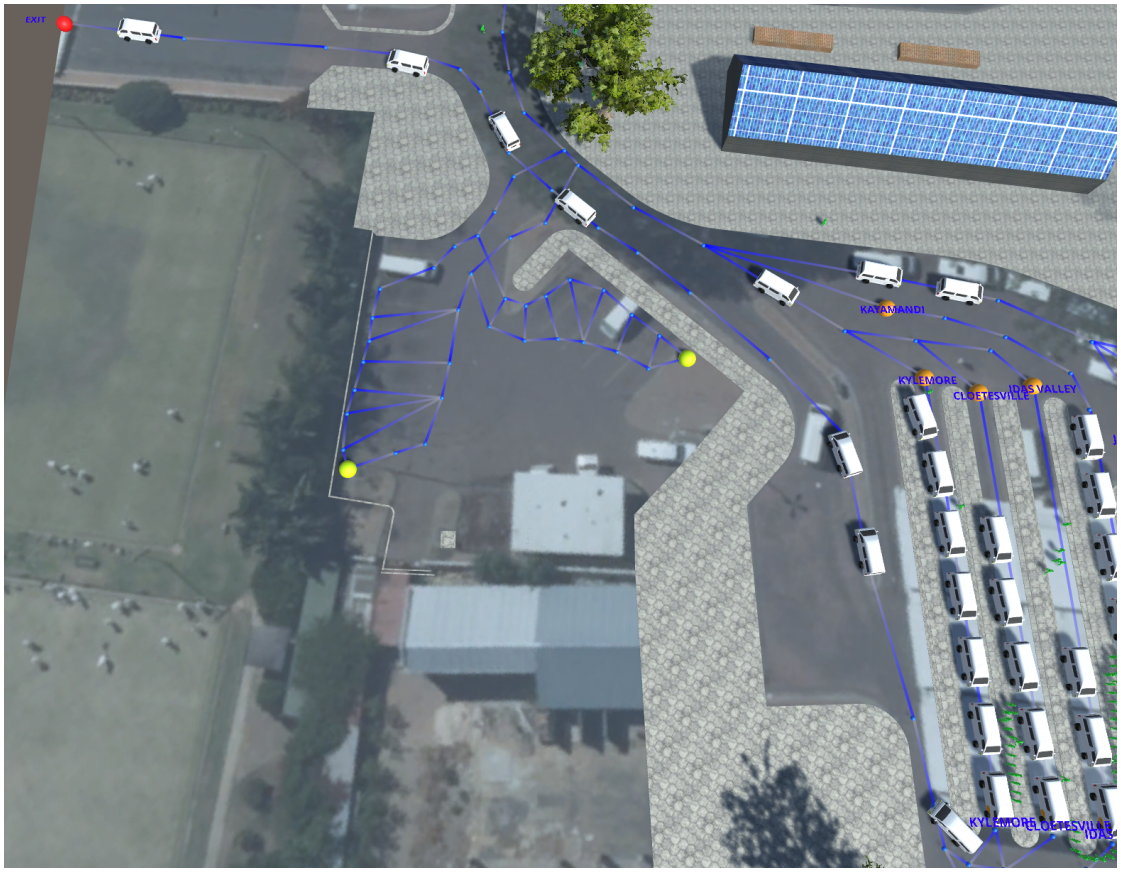


Figure 5.10: Taxis navigating to exit of rank

The taxi navigation system is deemed to function as intended.

Loading commuters

Before commuters can be loaded, they must find their way to an appropriate BayNode. To this, they must navigate through the rank geometry. Figure 5.11 shows that the commuter agents were able to navigate the rank geometry. They walked around walls and bollards and walked toward the appropriate BayNodes.

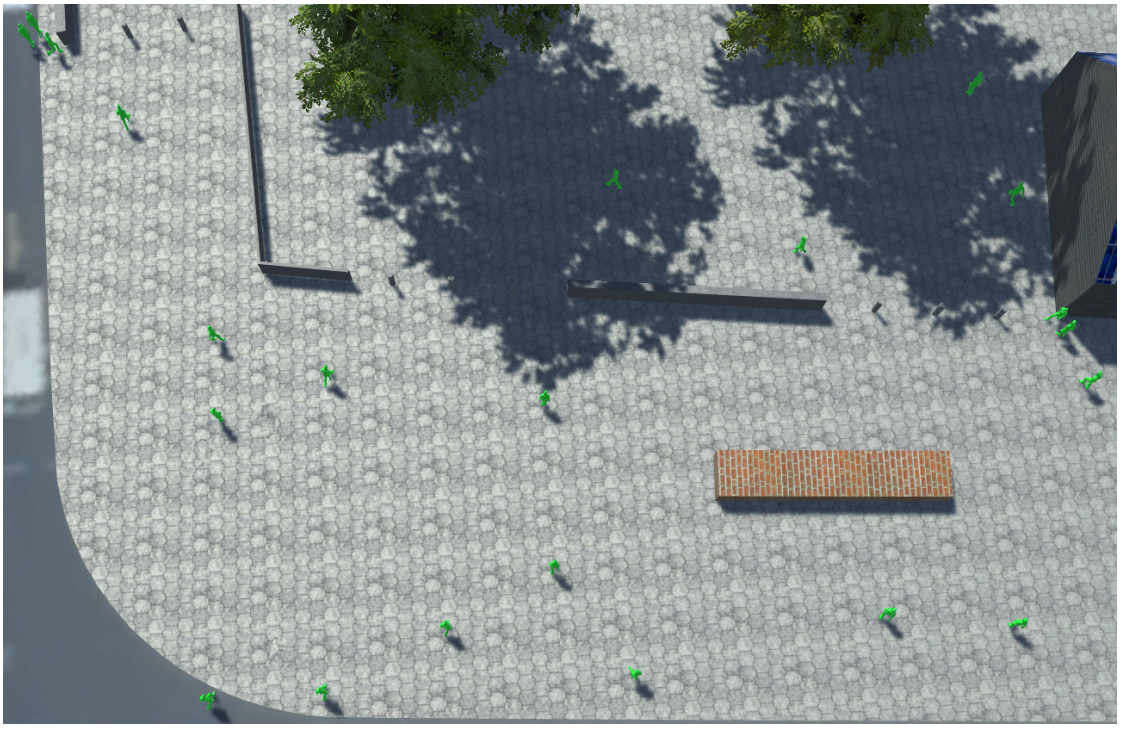


Figure 5.11: Commuters navigating the rank geometry

Commuter agents must also be able to navigate around and between taxi agents. Figure 5.12 shows that the commuters were indeed able to do this. Figure 5.12 also shows that the commuters were able to form queues on the parallel islands as was desired.

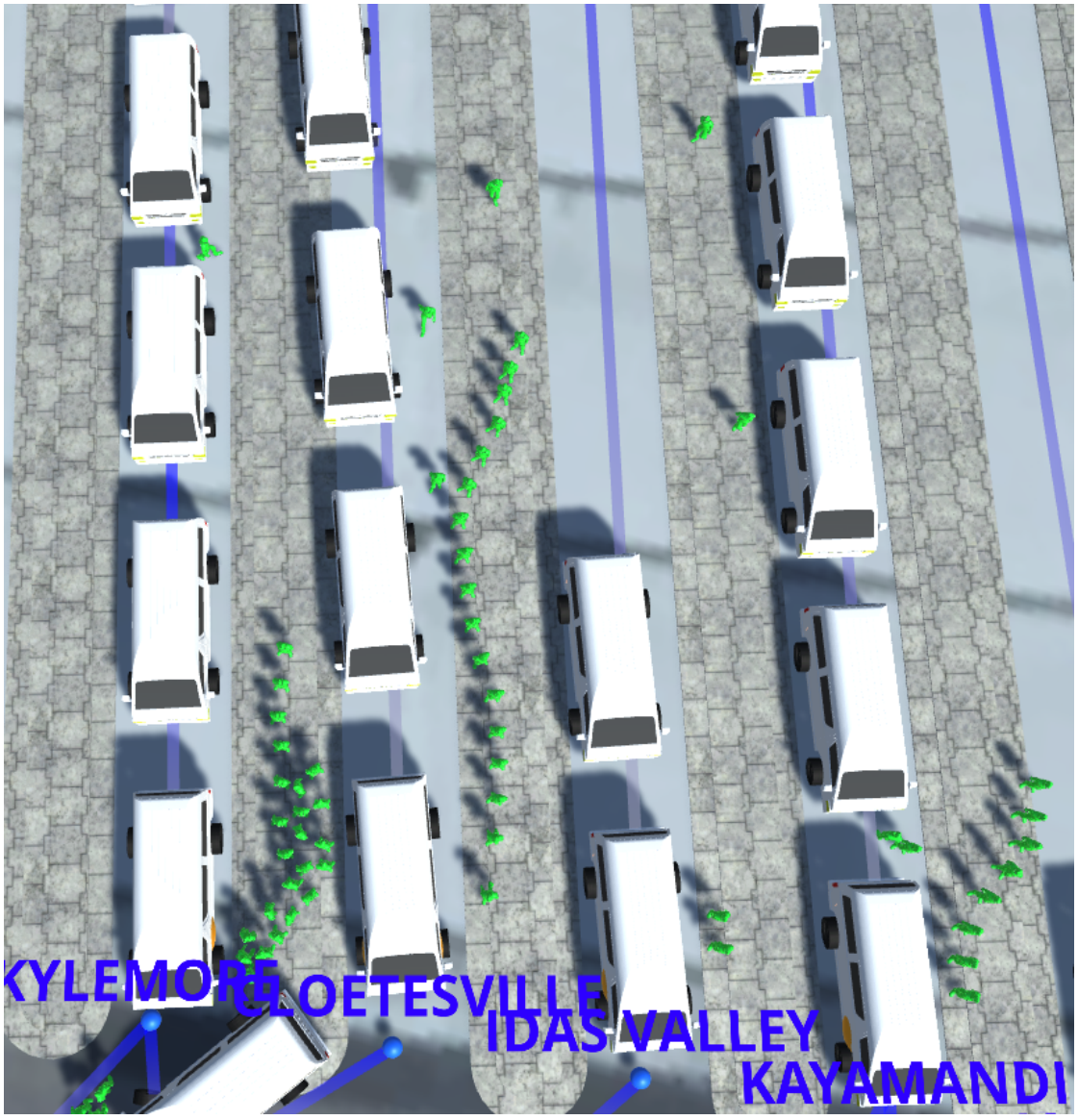


Figure 5.12: Commuters navigating between taxis and queuing

Finally, the commuter agents were also seen to be loaded into the taxis. This was visually represented by having the first commuter in the queue of commuters at a BayNode disappear after the desired time delay. Figure 5.13 shows commuter agents that want to go to Kayamandi being loaded into a taxi whose destination is Kayamandi at the BayNode whose corresponding destination is Kayamandi.

Figure 5.13: Commuters being loaded into a taxi

The system for loading commuters in ACTS is deemed to function as intended.

Taxi parking

When an aisle to a BayNode becomes full, it was desired that any taxis arriving go to park in a separate waiting area. During the testing of ACTS, it was observed that the aisle leading to the Kayamandi BayNode became saturated most often. Figure 5.14 shows the parking system functioning as intended. The Kayamandi aisle has become saturated and the taxi agents arriving to go there instead park at an adjacent holding area. When a position in the

Kayamandi aisle opens up, the first taxi agent that parked starts driving again and joins the back of the queue of taxis in the aisle.

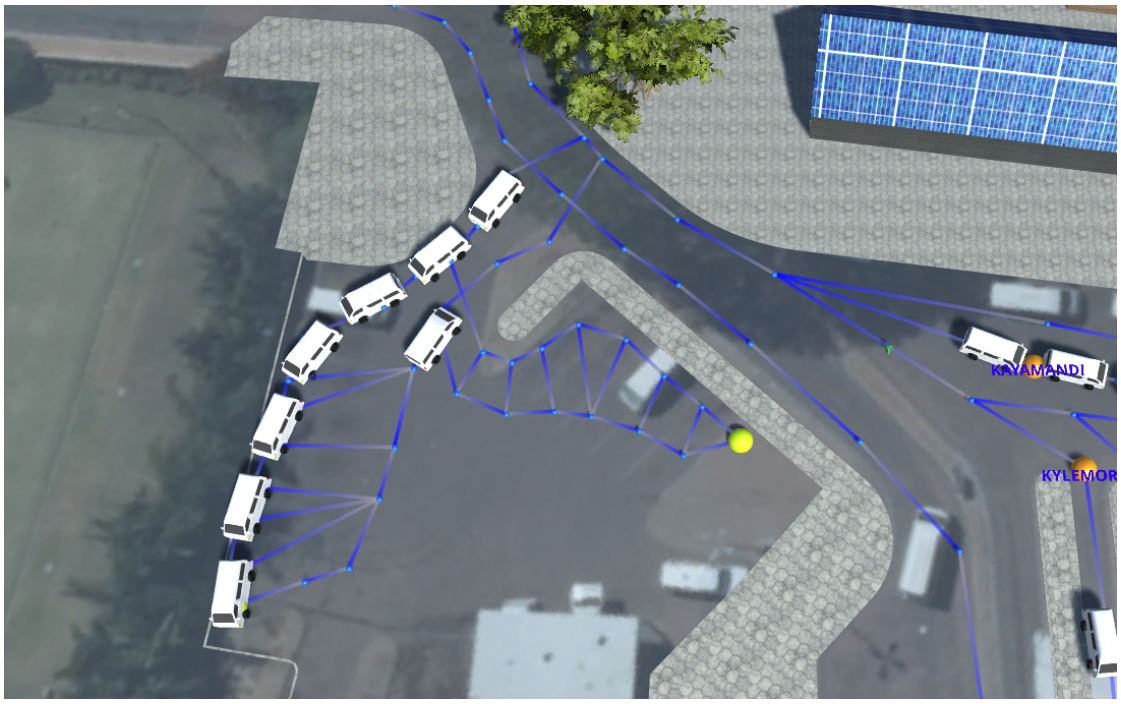


Figure 5.14: Taxis performing the parking action

The parking system was seen to be functioning as intended.

Taxi arrivals

It was desired that taxi agents arrive throughout a simulation at the times determined as per section 4.7. It was also desired that they drive into the entrance of the taxi rank where the design of the taxi rank allows for them to do so. When ACTS is run, the taxi agents do indeed arrive at the desired times. Figure 5.15 shows that they also use the entrance of the taxi rank as intended.

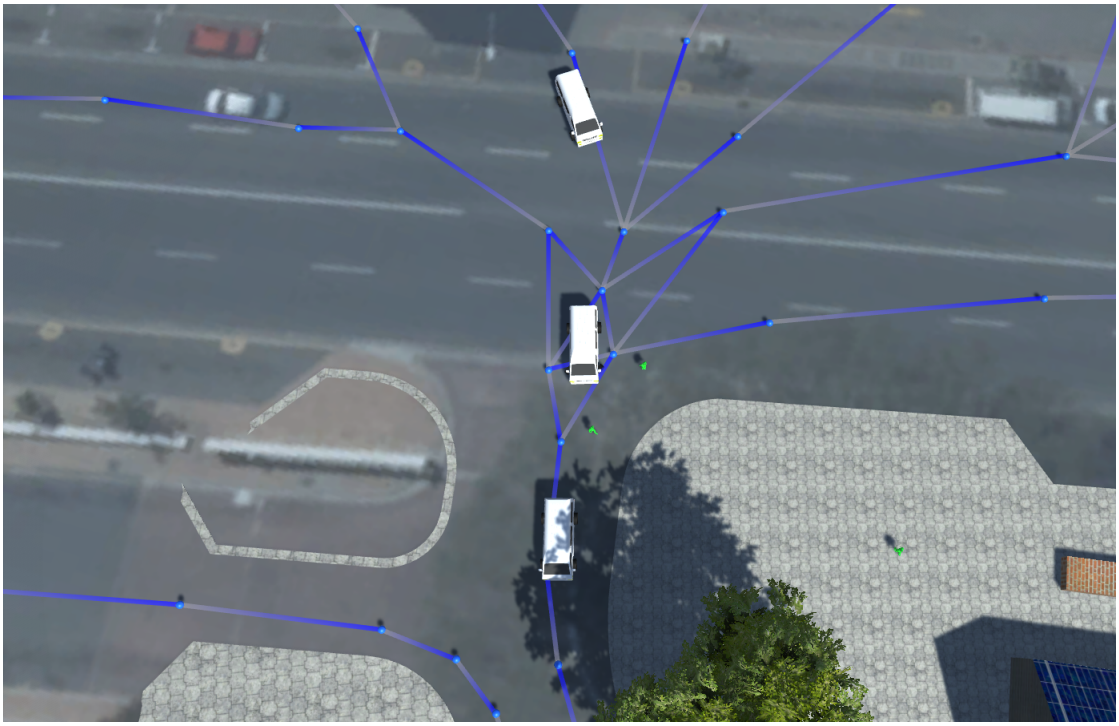


Figure 5.15: Taxis arriving at the rank

The system governing the arrival of the taxi agents is deemed to function as intended.

Commuter arrivals

There are two ways for commuters to arrive at the taxi rank. They either approach the rank on foot or they disembark from a taxi agent. When arriving on foot, it was desired that their inter-arrival times be determined as discussed in 4.7. The inter-arrival times of the commuter agents were observed to indeed follow the intended values. When there were more taxis in the rank, the inter-arrival times became shorter. When there were fewer taxis in the rank, the inter-arrival times became longer. This matches what was expected. Figure 5.16 shows commuters arriving and entering the rank on foot.

Figure 5.16: Commuters arriving outside the taxi rank

When commuters arrive at the rank on a taxi, they are unloaded and then proceed to walk out of the rank. Figure 5.17 shows that ACTS produces the expected behaviour.

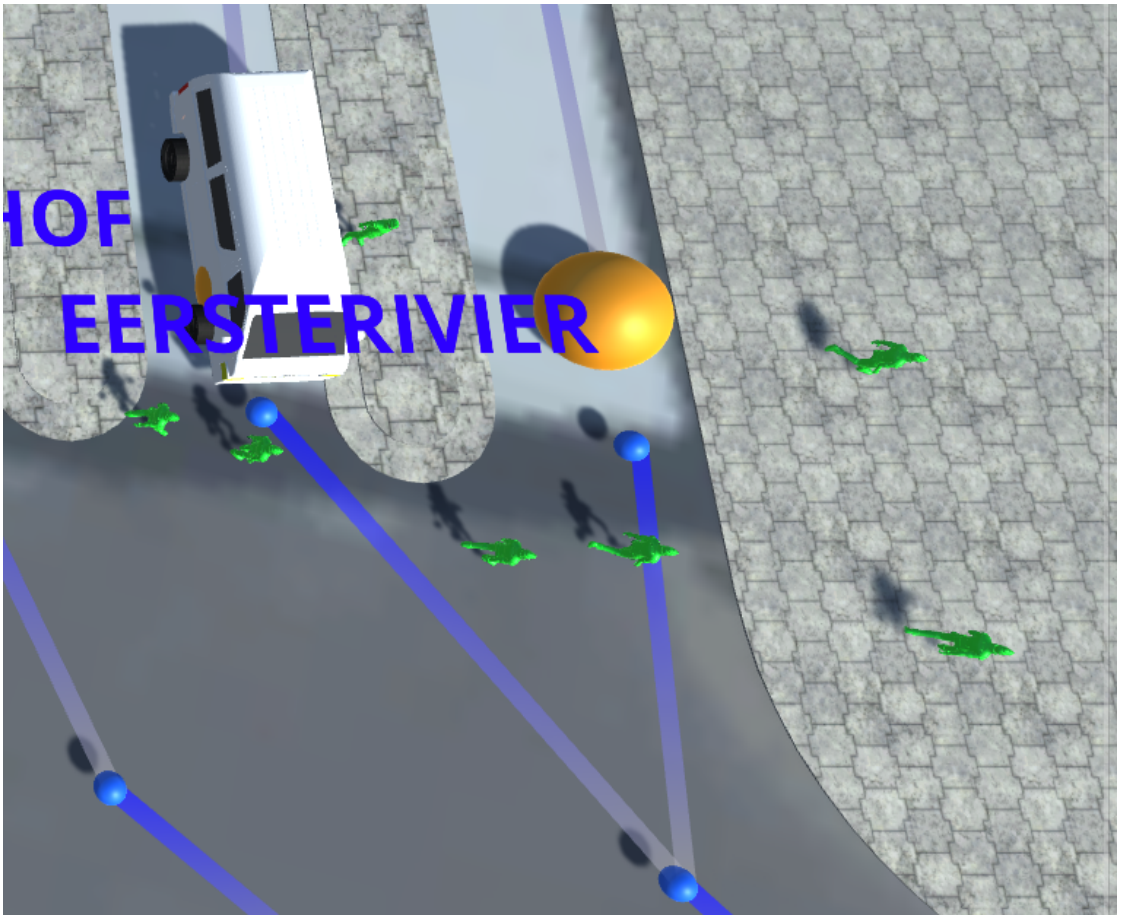


Figure 5.17: Commuters leaving the taxi rank after arriving on a taxi

Taxi sensing

The taxi sensing algorithm as described in 4.7 is intended to ensure that taxi agents maintain safe distances between each other as well as resolve right of way conflicts. Figure 5.18 shows that the algorithm is working as intended while the taxi agents are moving. The thin blue lines projecting forward out of the taxi agents represents the sensor length at which they will start applying their brakes.

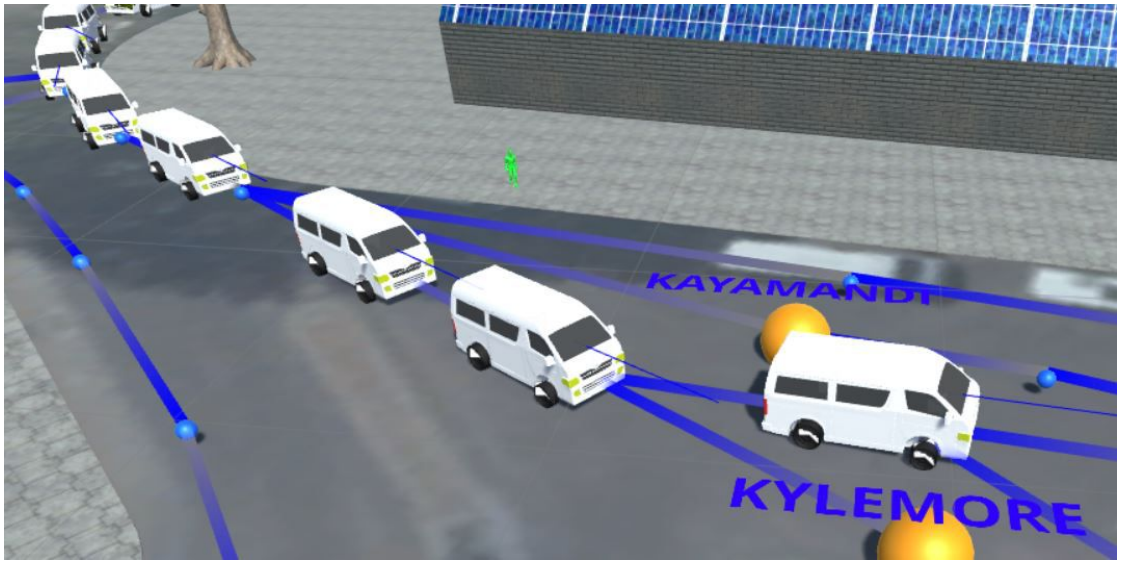


Figure 5.18: Taxis keeping a following distance while driving

Figure 5.19 shows that the taxi agents also maintain appropriate distances between each other while stopped. The distances are wide enough for commuter agents to navigate through them.

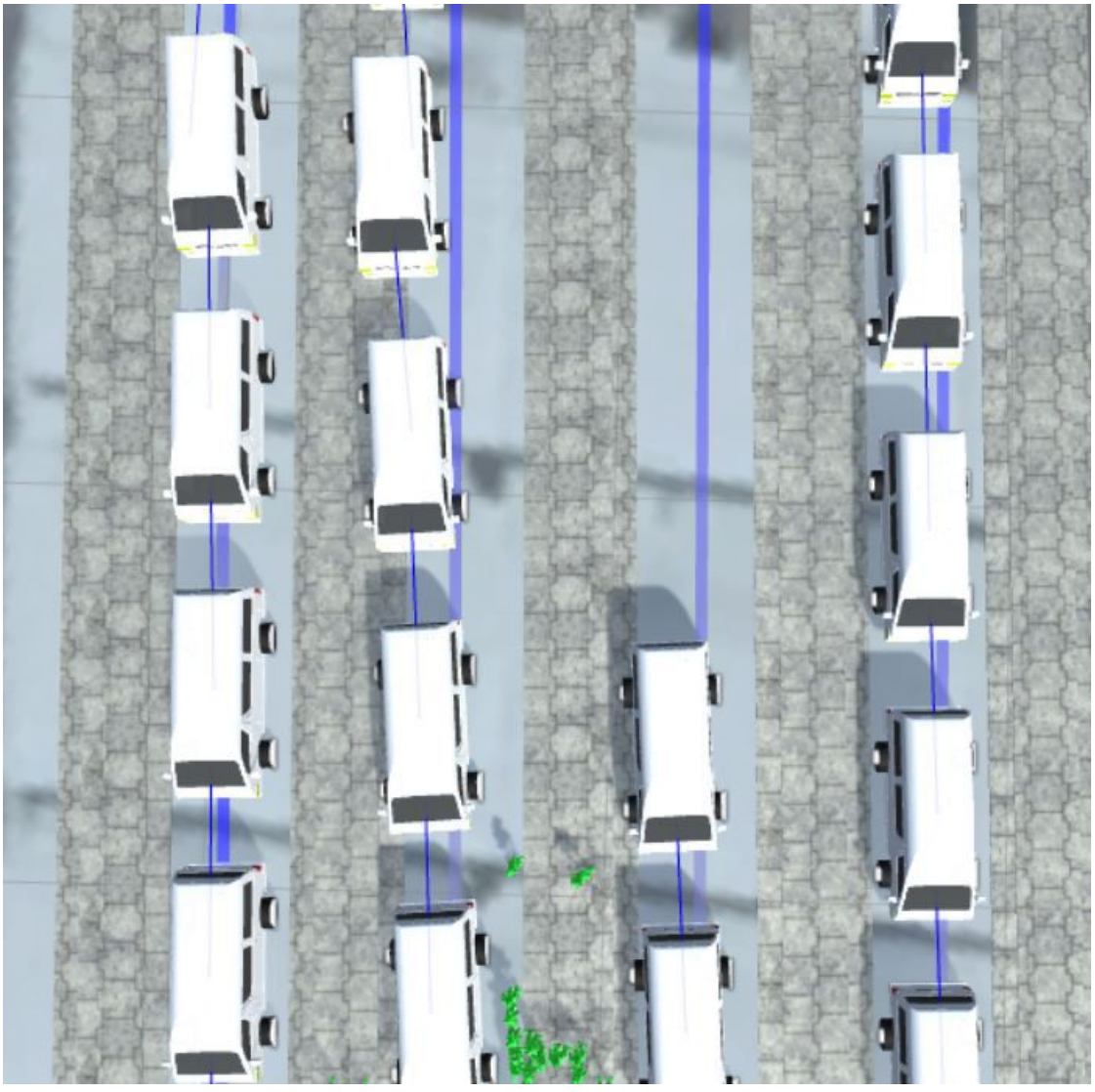


Figure 5.19: Taxis keeping distance between each other while stopped

The algorithm described in section 4.7 requires that the taxi agents project their sensors in the direction of their steering angle. This ensures that appropriate following distances are also observed around corners. Figure 5.20 shows that this indeed does occur in ACTS.

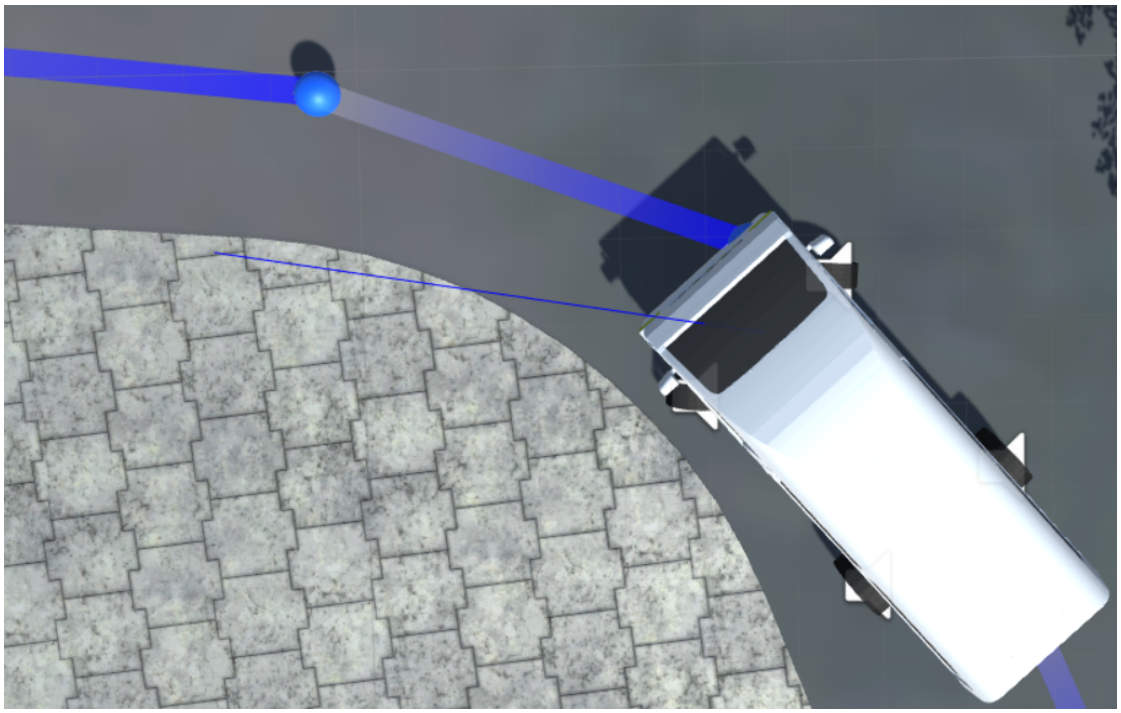


Figure 5.20: Taxi projecting sensor around corner

The taxi agent sensing system is deemed to function as intended.

5.4 Visualisation and interpretation of numeric output

The main numerical output produced by ACTS is a record of what times the taxi agents leave during a simulation. The times that each taxi arrived at the rank as well as the times that each taxi agent was observed to leave is stored as well. The unit that the times are stored in is hours past midnight. Three different types of plots were made of the data, each providing its own insights into the validity of ACTS:

- Histograms of AM, PM and Saturday peak time departures. Separate figures were made for observed and predicted values.
- Scatter plots that have the observed departures on the horizontal axis and ACTS predicted departures on the vertical axis. Such scatter plots were prepared for each of the three peak periods studied.
- Box and whisker plots for each of the following data sets:
  - Observed arrivals
  - Observed departures
  - Four sets of predicted departures. The four sets were chosen randomly from the sets produced by all the simulation runs completed.

Histograms of output

AM peak period

The histograms of the AM peak period observed departure times as well as the AM peak period predicted departure time can be seen in figure 5.21. The bins are fifteen minutes wide.

The distribution of the observed data and the data predicted by ACTS is shown to be remarkably similar. Both show that the rank is busiest between 07:15 and 07:30 and then again from 08:00 to 08:15. The differences between the two distributions are found at the extreme ends: ACTS predicts traffic before 06:30 as well as traffic after 09:00. The left discrepancy is interpreted to mean that ACTS slightly overestimates the number of commuters that arrive before 06:30 and that this leads to taxis departing earlier than they would have in reality. The two vehicle discrepancy is however not thought to have a large effect on the overall results. The discrepancy on the right extreme ends of the histograms can be explained by the fact that the survey conducted at Bergzicht taxi rank stopped making observations after 09:00 even though there were still taxis in the rank. In ACTS, those taxis were still allowed to leave the rank and have their departures recorded. It is therefore found to be justifiable to have departures after 09:00 in the predicted results.

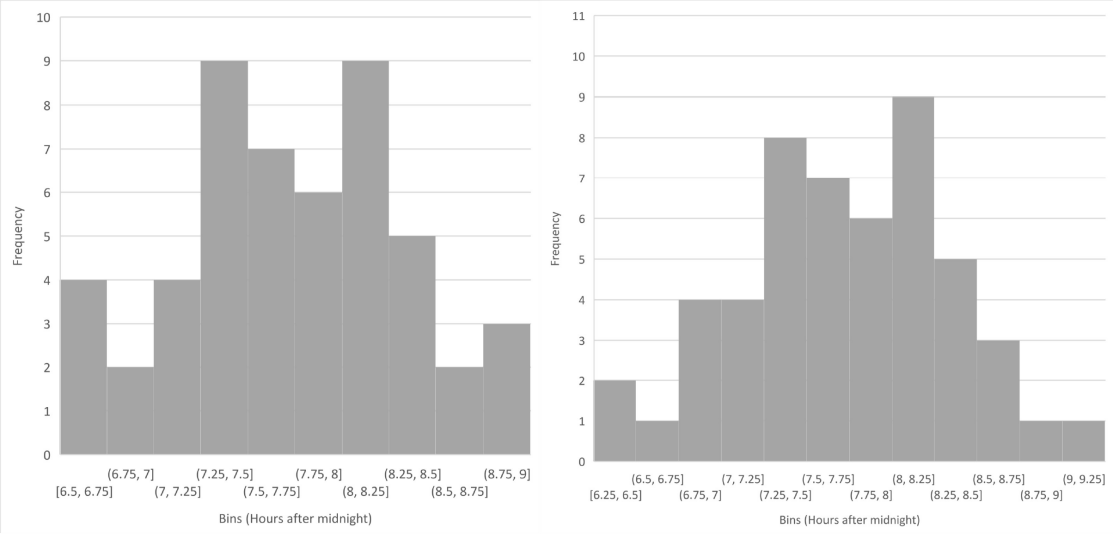


Figure 5.21: Histogram showing the AM peak observed departures (Left) and predicted departures (Right)

PM peak period

The histograms of the PM peak period observed departure times as well as the PM peak period predicted departure time can be seen in figure 5.22. The bins are fifteen minutes wide.

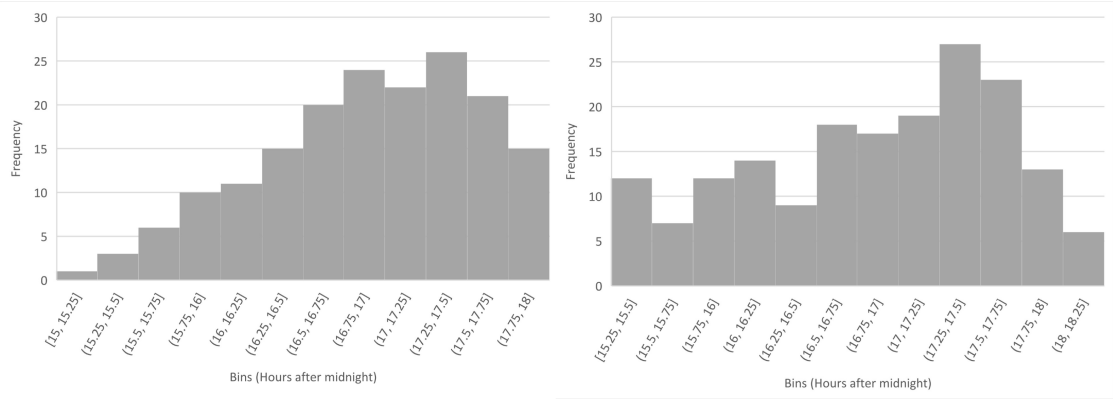


Figure 5.22: Histogram showing the PM peak observed departures (Left) and predicted departures (Right)

The distribution of the observed data and the data predicted by ACTS is shown to be similar. Both are left skewed, interpreted to mean that the rank is busier toward the end of the survey period. They both have their peak in the same bin and show that the rank is at its busiest between 17:15 and 17:30.

Both distributions also show that the rank becomes less busy almost immediately after the peak bin. The major difference between the two distributions is that ACTS predicts a frequency of twelve taxis between 15:15 and 15:30 while a frequency of three was observed. After this bin however, they both predict a frequency of six between 15:30 and 15:45. The discrepancy at the left extreme of the distribution is believed to arise from the manner in which the observations were made. The survey was conducted from 15:00 until 18:00. Any vehicles that were already in the rank were recorded as having arrived at 15:00 even though they had arrived earlier. In ACTS, these vehicles were scheduled to arrive as they had been recorded: At 15:00. As the taxis in reality had arrived earlier, they would also leave earlier than the agents in ACTS. Forty of the observations recorded taxis as arriving at 15:00. It is believed that this explains why there was an abnormally high number of departures predicted fifteen minutes later.

Saturday peak period

The histograms of the Saturday peak period observed departure times as well as the Saturday peak period predicted departure time can be seen in figure 5.23. The bins are fifteen minutes wide.

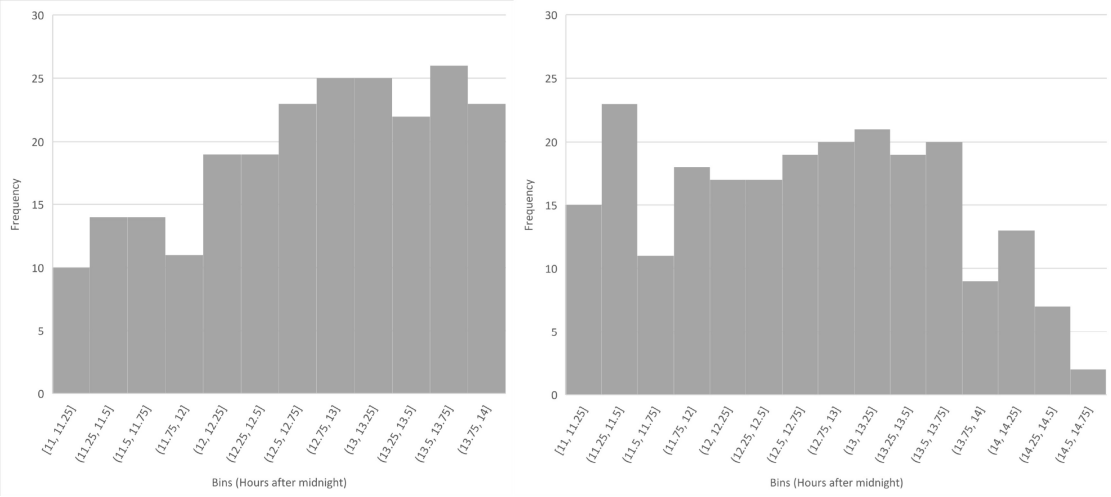


Figure 5.23: Histogram showing the Saturday peak observed departures (Left) and predicted departures (Right)

The distributions of the observations and predictions are similar in that they both have a relatively flat peak between 12:30 and 13:45. ACTS however predicts that the peak is slightly lower than was observed. ACTS also predicts that the rank is much busier between 11:00 and 11:30 than was observed. In addition, ACTS predicts departures after 14:00. The departures after 14:00

can be explained by the fact that there were still taxis in the rank at 14:00 when the survey was stopped. The higher frequency of predictions at the left extreme is once again believed to be the result of the survey recording that taxis already in the rank had arrived at the start of the survey period, 11:00, even though they had been there earlier. Nineteen of the observations recorded taxis as arriving at 11:00. During the simulation runs, it was observed that traffic jams tended to form during the peak time in ACTS. In reality, the taxi drivers make better use of any available space and they therefore mitigate the onset of traffic jams. Traffic jams lowered the throughput of taxi agents in ACTS, and it is believed that this is the reason that the distribution peak was lower. It was observed that the traffic jams were usually a result of the Kayamandi aisle becoming saturated. This has the result of clogging the bottleneck that is the entrance to the rank. The rest of the rank operates well under capacity when the Kayamandi aisle starts causing traffic jams.

Scatter plots of output

AM peak period

Figure 5.24 is a scatter plot showing the correlation between the observations made during the taxi rank survey and the predictions produced by ACTS for the AM peak time. Ideally, the slope of the fitted line should be unity. There should also be as little variance about the fitted line as possible. Each cross on the figure represents the departure of a taxi. For the AM peak period, it was determined that the slope of a fitted line is 0.8411 and the R^2 value 0.744. These results indicate that ACTS should not be used to make predictions about individual taxis during an AM peak period. This is believed to stem from the variability and unpredictability of the system being modelled.

PM peak period

Figure 5.25 is a scatter plot showing the correlation between the observations made during the taxi rank survey and the predictions produced by ACTS for the PM peak time. For the PM peak period, it was determined that the slope of a fitted line is 1.1375 and the R^2 value 0.8594. These results indicate that ACTS should not be used to make predictions about individual taxis during a PM peak period. This is believed to stem from the variability and unpredictability of the system being modelled.

Saturday peak period

Figure 5.26 is a scatter plot showing the correlation between the observations made during the taxi rank survey and the predictions produced by ACTS for the Saturday peak time. For the Saturday peak period, it was determined that the slope of a fitted line is 1.0347 and the R^2 value 0.8093. The slope of the

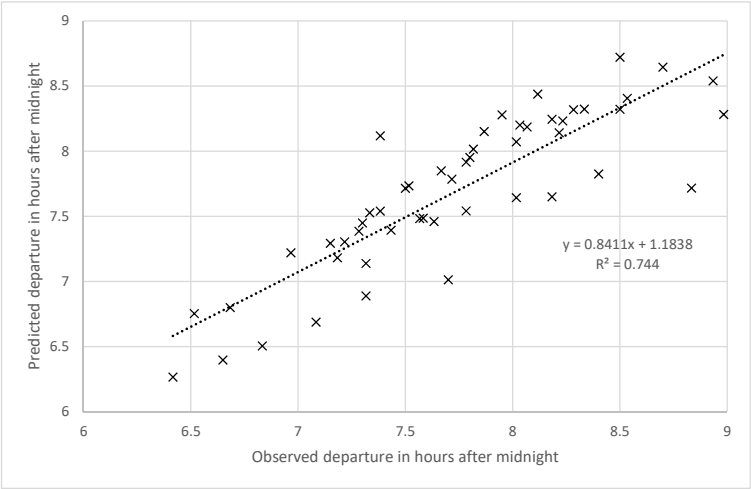


Figure 5.24: Scatter plot showing the correlation between the observed and predicted departure times of the AM peak period

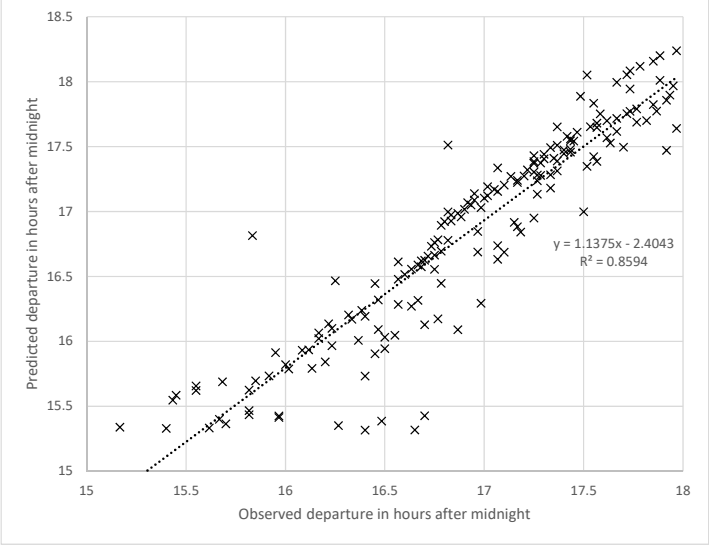


Figure 5.25: Scatter plot showing the correlation between the observed and predicted departure times of the PM peak period

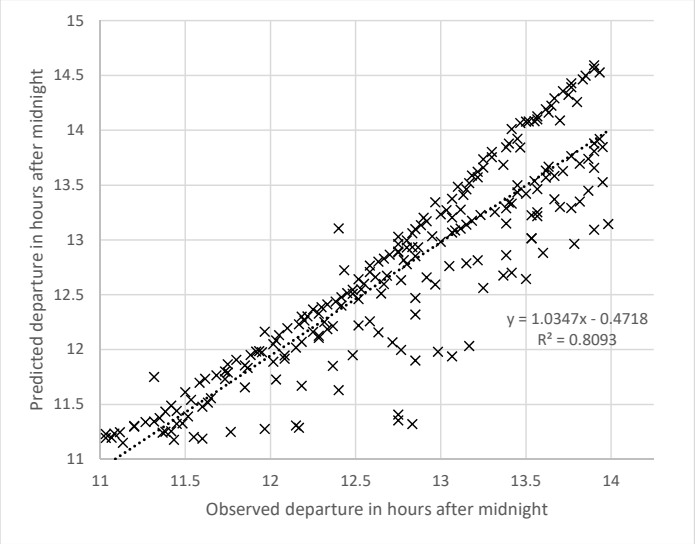


Figure 5.26: Scatter plot showing the correlation between the observed and predicted departure times of the Saturday peak period

fitted line is very close to one and many of the points do follow it. There are also many points that do not follow the fitted line. Indeed, there is an entire series of points that seem to be following a steeper trend than what was desired. This steeper gradient indicates that ACTS predicts longer waiting times than those that actually ocurred. The steeper gradient that these points follow is believed to be a result of the traffic jams that occur in ACTS toward the end of the Saturday peak period. The variability in the results as well as the slope deviation toward the end of the peak period indicate that ACTS should not be used to make predictions about individual taxis during a Saturday peak period.

Box and whisker plots of output

AM peak period

Figure 5.27 shows six box and whisker plots concerning the AM peak period. The first from the left shows the distribution of the arrivals at the taxi rank. The second plot from the left shows the distribution of the observed departure times. The four plots that follow show the distributions produced by four different simulation runs. The vertical axis represents the number of hours after midnight.

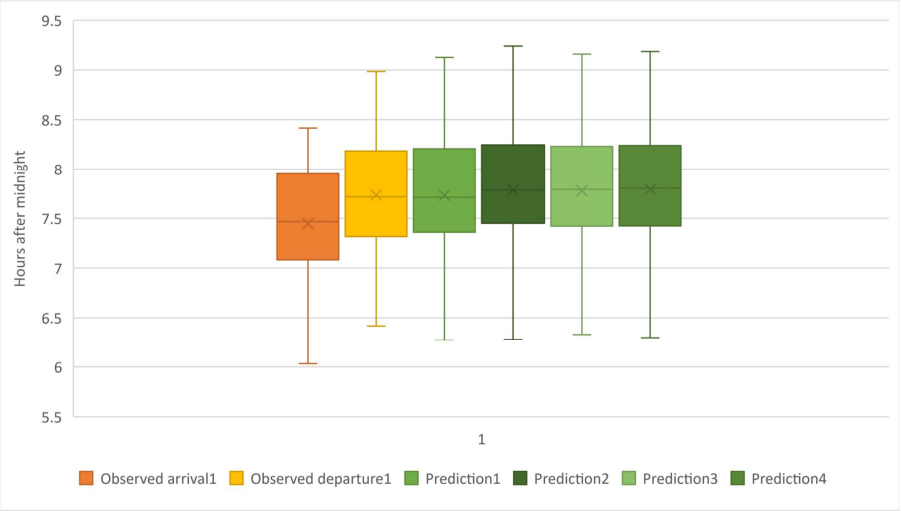


Figure 5.27: Box and whisker plots for AM peak arrivals, observed departures and the departures predicted by four simulation runs

The arrivals plot is shifted lower than the rest of the plots. This makes sense as a departure must surely always be after an arrival. It is observed that the means and medians of the various predictions are similar to each other and to the observed means and medians. The simulation runs also provide values for the lower and upper quartiles that are similar to the lower and upper quartiles of the observed departure times. At the upper extreme of the plots, the results of the simulation runs can be seen extending past 09:00 while the observed departure times do not. This is a result of the survey ending at 09:00. The lower extreme of the plots shows that the results of the simulations extend to a slightly earlier time than those that were observed. This indicates that ACTS predicts there to be slightly more commuters arriving on foot at the rank between 06:00 and 06:30 than what must have occurred at that time interval in reality.

PM peak period

Figure 5.28 shows six box and whisker plots concerning the PM peak period. The first from the left shows the distribution of the arrivals at the taxi rank.

The second plot from the left shows the distribution of the observed departure times. The four plots that follow show the distributions produced by four different simulation runs. The vertical axis represents the number of hours after midnight.

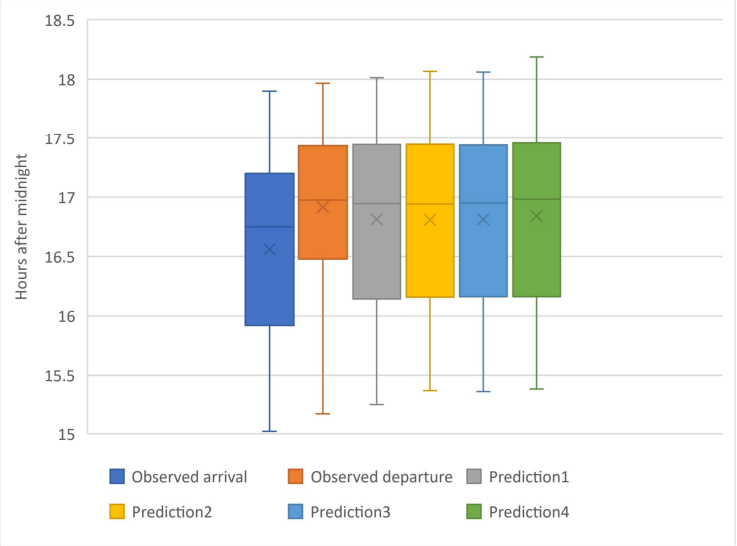


Figure 5.28: Box and whisker plots for PM peak arrivals, observed departures and the departures predicted by four simulation runs

The arrivals plot is shifted lower than the rest of the plots. This makes sense as a departure must surely always be after an arrival. It is observed that the means and medians of the various predictions are similar to each other and to the observed means and medians. The simulation runs also provide values for the upper quartiles that are similar to the upper quartiles of the observed departure times. The lower quartiles predicted by the simulation runs are consistently lower than what was observed. This indicates that ACTS predicts more departures in the earlier part of the peak period than what occurred. It is believed that this is the result of having taxis already in the rank at 15:00 being recorded as arriving at 15:00. At the upper extreme of the plots, the results of the simulation runs can be seen extending past 18:00 while the observed departure times do not. This is a result of the survey ending at 18:00. The lower extreme of the plots shows that the results of the simulations extend to a slightly later time than those that were observed. This is because there were taxis already in the rank before 15:00 that did not spend much time after

15:00 there, while in ACTS, those taxis were scheduled to arrive at 15:00 and therefore left the rank later than they would have if they had arrived earlier.

Saturday peak period

Figure 5.29 shows six box and whisker plots concerning the Saturday peak period. The first from the left shows the distribution of the arrivals at the taxi rank. The second plot from the left shows the distribution of the observed departure times. The four plots that follow show the distributions produced by four different simulation runs. The vertical axis represents the number of hours after midnight.

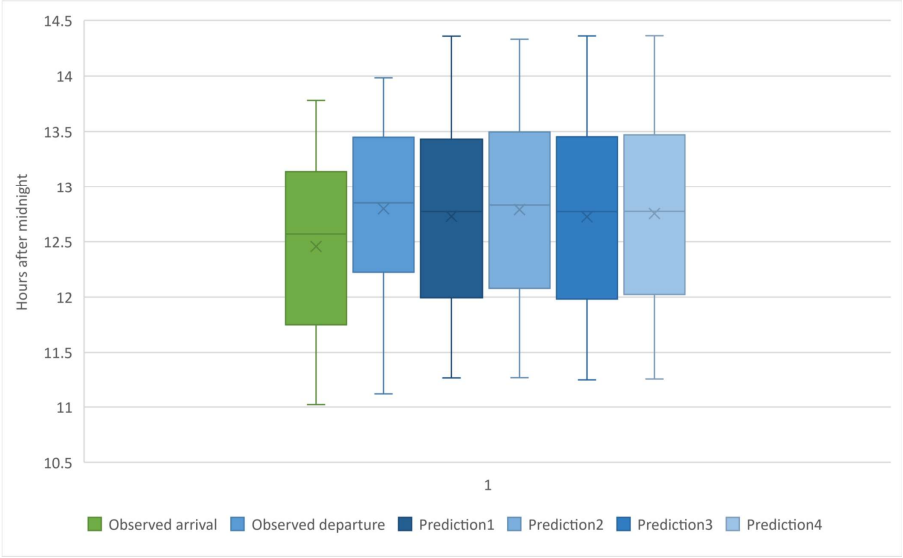


Figure 5.29: Box and whisker plots for Saturday peak arrivals, observed departures and the departures predicted by four simulation runs

As expected, the arrivals plot is shifted lower than the rest of the plots. This makes sense as a departure must surely always be after an arrival. It is observed that the means and medians of the various predictions are similar to each other and to the observed means and medians. The simulation runs also provide values for the upper quartiles that are similar to the upper

quartiles of the observed departure times. The lower quartiles predicted by the simulation runs are consistently lower than what was observed. This indicates that ACTS predicts more departures in the earlier part of the peak period than what occurred in reality. It is believed that this is the result of having taxis already in the rank at 11:00 being recorded as arriving at 11:00. At the upper extreme of the plots, the results of the simulation runs can be seen extending past 14:00 while the observed departure times do not. This is a result of the survey ending at 14:00. The lower extreme of the plots shows that the results of the simulations extend to a slightly later time than those that were observed. This is because there were taxis already in the rank before 11:00 that did not spend much time after 11:00 there, while in ACTS, those taxis were scheduled to arrive at 11:00 and therefore left the rank later than they would have if they had arrived earlier.

The box and whisker plots in this section provide confidence in the ability of ACTS to predict means, medians and upper quartiles. If taxis were scheduled to arrive at the rank before the start of the peak period, it is believed that ACTS would also be able to predict lower quartiles confidently.

5.5 Historical data validation

The tests conducted in section 5.4 indicated that ACTS may be useful for determining summary statistics for the operation of a taxi rank during a peak period. In this section, this ability is scrutinised with formal statistical tests. The summary statistic used as an example is the average amount of time that the taxis spend in a rank. The rank time of a taxi can be calculated as in equation 5.1:

$$t_r = t_d - t_a \tag{5.1}$$

Where:

- t_r is the rank time or amount of time that the taxi spends in the rank.
- t_d is the time that the taxi departed from the rank.
- t_a is the time that the taxi arrived at the rank.

If the rank time of each taxi in a peak period is summed and the sum is divided by the total number of taxis that departed from the rank, the average rank time $\bar{t}_r$ can be calculated. Two average rank times are of interest in this section, the average rank time calculated for observed departures and the average rank time calculated for departures predicted by ACTS. The average rank time calculated for observed departures will be referred to as μ_r. Because every ACTS simulation run yields slightly different results, it was decided that the mean of these results be used to represent the output of ACTS. The mean of the average

rank times predicted by ACTS will be referred to as $\overline{T}_r$. Before any tests were conducted, the following transformations were applied to the collected data:

- Any records where a taxi arrives exactly at the start of a peak period were not used. This is to avoid calculating rank times from erroneous data as any taxis already in a rank were recorded as having arrived at the start of the peak period even though they had arrived earlier.

- Any records that show that a taxi departs exactly at the end of a peak period were not used. This was done to avoid calculating rank times from erroneous data as any taxis that were in the rank at the end of the peak period were recorded as having left the rank at the peak period even though they departed later.

- For the AM and PM peak periods, any rank time less than five minutes or more than thirty minutes was not included in the analysis. This was done as any taxi spending less than five minutes in the rank is most probably not loading a full complement of passengers as was modelled. Any taxi spending more than thirty minutes in the rank is also most probably not there to load passengers.

- For the Saturday peak period, any rank time less than five minutes or more than sixty minutes was not included in the analysis. This was done as any taxi spending less than five minutes in the rank is most probably not loading a full complement of passengers as was modelled. On a Saturday, it is less critical for passengers to leave as early as possible. This is because most trips on a Saturday are not trips to work. It is believed therefore that taxis would be willing to wait longer times for passengers to arrive and embark.

More than one hundred simulation runs were conducted for each peak period to collect data to calculate $\overline{T}_r$.

Hypothesis testing

Step 1: Statement of the null hypothesis

The null hypothesis is that the mean rank time observed for a peak period is equal to the mean rank time predicted by ACTS. The null hypothesis is shown formally in equation 5.2:

$$H_0 : \mu_r = \overline{T}_r \tag{5.2}$$

Step 2: Statement of the alternative hypothesis

The alternative hypothesis is that the mean rank time observed for a peak period is not equal to the mean rank time predicted by ACTS. The alternative hypothesis is shown formally in equation 5.3:

$$H_A : \mu_r \neq \overline{T}_r \tag{5.3}$$

Step 3: Statement of the significance level and control of errors

The significance level that will be used is $\alpha = 0.05$. This implies a confidence level of 95% in the test. The significance level is the likelihood of rejecting the null hypothesis when it is true. This is known as a type I error. In the field of simulation models, it is known as "model builder's risk" (Sargent 2013). Another more critical error for simulation models is to fail to reject the null hypothesis when it is false. This is known as a type II error, symbolically represented by β. In the field of simulation models, this type of error is known as the "model user's risk". The value of β depends on the sample size, the effect size δ and α. The effect size is calculated as in equation 5.4:

$$\delta = \frac{|\overline{T}_r - \mu_r|}{\sigma} \tag{5.4}$$

Where σ is the standard deviation. The effect size and therefore β differ for each of the peak periods. The calculated effect sizes and β values are reported alongside the results of the hypothesis testing done for each of the peak periods. For testing ACTS, we were not concerned with proving that the means were exactly the same, the numerator of 5.4 was replaced with a tolerance of one minute such that effect size is determined as in equation 5.5:

$$\delta = \frac{1/60}{\sigma} \tag{5.5}$$

Step 4: Collection of relevant data

For the AM peak period the following values were calculated:

- $\mu_r = 0.2397$ hours or 14.4 minutes.
- $\overline{T}_r = 0.2299$ hours or 13.8 minutes.

For the PM peak period the following values were calculated:

- $\mu_r = 0.2627$ hours or 15.8 minutes.

- $\overline{T}_r = 0.2340$ hours or 14.0 minutes.

For the Saturday peak period the following values were calculated:

- $\mu_r = 0.3107$ hours or 18.64 minutes.
- $\overline{T}_r = 0.3099$ hours or 18.59 minutes.

Step 5: Calculation of test statistics

The one-sample, two sided Student t-test with a significance level 0.05 was used. The t-test is used as it allows the testing of a hypothesis when the standard deviation of the population is not known. As the observations made are for a sample of all peak periods, they cannot be used to determine the population standard deviation. A two sided test is conducted as it is of interest when $\overline{T}_r$ is either more than or less than μ_r. The one-sample t-test is appropriate when comparing a sample mean to the population mean. The one-sample t-statistic can be calculated with equation 5.6:

$$t = \frac{\overline{X} - \mu}{s/\sqrt{n}} \tag{5.6}$$

Where:

- $\overline{X}$ is the sample mean.
- μ is the population mean.
- s is the standard deviation of the sample.
- n is the sample size.

The appropriate values were substituted into equation 5.6 as shown in equation 5.7:

$$t = \frac{\overline{T}_r - \mu_r}{s/\sqrt{n}} \tag{5.7}$$

Where:

- $\overline{T}_r$ is the mean average rank time produced by ACTS.
- μ_r is the mean observed average rank time.
- s is the standard deviation of the average rank times produced by ACTS.
- n is the number of average rank times produced by ACTS. This is the same as the number of simulations conducted.

If a t-statistic exceeds the critical value for a given significance level α and degrees of freedom, the null hypothesis is to be rejected. Another important factor in hypothesis testing is the "p-value". The p-value is the probability of obtaining a test result that is at least as extreme as the critical value given that the null hypothesis is true. If the p-value is less than the significance level α then there is significant evidence to reject the null hypothesis. If the p-value is more than α then there is not enough evidence to reject the null hypothesis. The t-statistic as well as corresponding p-values were calculated for each of the peak periods:
For the AM peak period, the following results were obtained:

- critical t value = $+-1.977$
- t-statistic = -5.880
- p-value ≈ 0.0000
- δ = 0.8426
- $\beta \approx 0.0\%$
- n = 140

For the PM peak period, the following results were obtained:

- critical t value = $+-1.980$
- t-statistic = -15.740191
- p-value ≈ 0.0000
- δ = 0.8394
- $\beta \approx 0.0\%$
- n = 118

For the Saturday peak period, the following results were obtained:

- critical t value = $+-1.984$
- t-statistic = 0.423
- p-value = 0.6731
- δ = 0.8412
- $\beta \approx 0.0\%$
- n = 100

Step 6: Interpretation of results

For the AM peak period, the t-statistic exceeds the critical t value. The p-value is also less than the specified α. There is therefore significant evidence to reject the null hypothesis H_0 and it is concluded that the alternative hypothesis $H_A : \mu_r \neq \overline{T}_r$ is true. This means that the mean predicted by ACTS is not the same as what was observed for the AM peak period.

For the PM peak period, the t-statistic exceeds the critical t value. The p-value is also less than the specified α. There is therefore significant evidence to reject the null hypothesis H_0 and it is concluded that the alternative hypothesis $H_A : \mu_r \neq \overline{T}_r$ is true. This means that the mean predicted by ACTS is not the same as what was observed for the PM peak period.

For the Saturday peak period, the t-statistic is less than the critical t value. The p-value is also more than the specified α. There is therefore not significant evidence to reject the null hypothesis H_0. Given that β approaches zero, there is high confidence in the result that ACTS predicts the correct average rank time for Saturdays. Justification for the low β values can be found in figure 5.30. The figure is a plot of the power of a hypothesis test as a function of the number of samples taken. The power of a test is the complement of its β: Test power = $1 - \beta$. From the figure, it can be seen that for a test where the effect size es is 0.84, about thirty five samples are needed to achieve a power approaching one. As more than one hundred simulation runs were conducted for each peak period, the power of the hypothesis tests conducted approaches one and therefore, β approaches zero.

The fact that the null hypothesis was rejected for the AM and the PM peak period tests, is not discouraging. It was expected that the model would not fully be able to recreate the exact circumstances that played out during the peak periods surveyed. In addition, only one survey per peak period was available. If more surveys become available, the two means may yet converge for the AM and PM peak periods. In addition, it was observed that even though the means could not proven to be equal, they were very close to each other. The slight error may yet be acceptable. To quantify the error for each peak period, confidence interval testing was employed.

Confidence interval testing

Confidence interval testing is concerned with whether the results of simulations are close enough to the real system values. The confidence interval for μ_r is shown in equation 5.8:

$$CI = \left[\overline{T}_r - t_{\frac{\alpha}{2},n-1}, \overline{T}_r + t_{\frac{\alpha}{2},n-1}\right] \tag{5.8}$$

For this section, an α of 0.05 will be used. This implies that there is a 95% probability that the confidence interval contains the true mean of the predicted

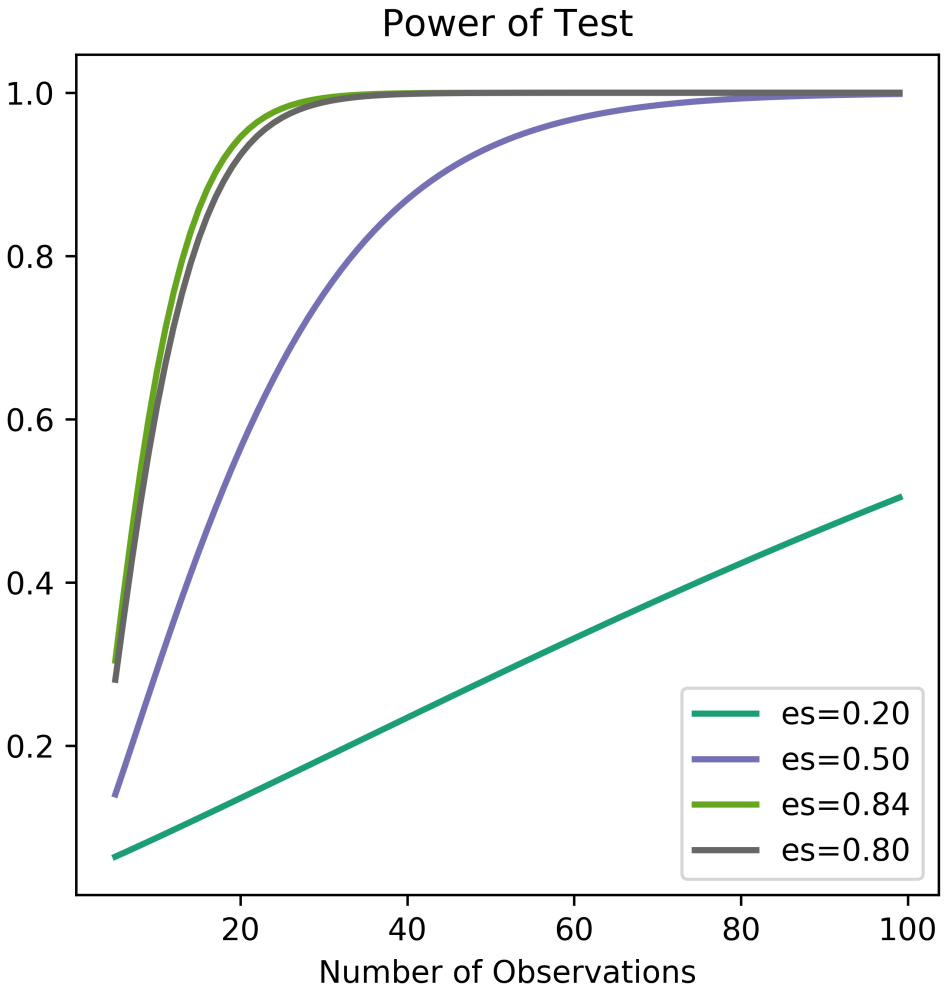


Figure 5.30: Power of the hypothesis test conducted as a function the number of samples

CI does not contain μ	**CI contains μ**
If the best case error $> \varepsilon$ The model needs to be refined	If the best case error $> \varepsilon$ then more simulations needed
If best case error $\leq \varepsilon$ then more simulations needed	If the worst case error $> \varepsilon$ then more simulations needed
If the worst case error $\leq \varepsilon$ then accept the model	If the worst case error $\leq \varepsilon$ then accept the model

Table 5.1: Conditions for accepting simulation model as indicated by results of confidence interval testing

average waiting times if an infinite number of simulations were conducted. The best case error is the distance between μ_r and the closest bound of CI to μ_r. The worst case error is the distance between μ_r and furthest bound of CI to μ_r. For confidence interval testing, the acceptable error ε is defined such that ε is small enough that valid decisions can be made from the results of simulations. (Günes 2012) provides table 5.1 to help decide whether to accept or reject a simulation model.

Confidence interval testing results

For the AM peak period, the following results were obtained:

- $\mu_r = 0.2397$ hours or 14.38 minutes
- 95% CI = $\left[0.2296, 0.2302\right]$ in terms of hours or $\left[13.78, 13.81\right]$ in terms of minutes
- Best case error = 0.009548 hours or 0.5729 minutes
- Worst case error = 0.01011 hours or 0.6066 minutes

For the PM peak period, the following results were obtained:

- $\mu_r = 0.2627$ hours or 15.76 minutes
- 95% CI = $\left[0.2336, 0.2343\right]$ in terms of hours or $\left[14.02, 14.06\right]$ in terms of minutes
- Best case error = 0.02843 hours or 1.706 minutes
- Worst case error = 0.02911 hours or 1.746 minutes

For the Saturday peak period, the following results were obtained:

- $\mu_r = 0.3099$ hours or 18.59 minutes

- 95% CI = $\left[0.3103, 0.3111\right]$ in terms of hours or $\left[18.62, 18.67\right]$ in terms of minutes
- Best case error = 0.0004412 hours or 0.02647 minutes
- Worst case error = 0.01235 hours or 0.07413 minutes

Interpretation of results

The narrow nature of the confidence intervals produced is a result of enough simulation runs being conducted that the predicted mean can be confined to narrow bounds. The worst error seen over all of the results is 1.746 minutes for the PM peak period. This is approximately 9% of the expected value. The best case error seen over all of the results is 0.02647 minutes for the Saturday peak period. This is approximately 0.1% of the expected value. It is believed that the worst case error is smaller than what would be selected as an appropriate ε for all peak periods. It is therefore argued that the ACTS model should be accepted as part of a decision making process that considers the average time that taxis spend in a taxi rank.

Chapter 6

Conclusion

6.1 Research outcomes

The goal of this research project was to develop a methodology that could provide answers to "what if?" questions posed about minibus-taxi loading facilities. It was proposed that the development of a simulation model would provide a way for designers to test the effects of their ideas without going to the expense or risk of building a minibus-taxi loading facility. In section 1.4 the following sequence of actions was decided upon:

1. Report on the broad context of the informal public transport sector in South Africa
2. Report on the role that the minibus taxi loading facility plays in this context
3. Report on the current methods used to design minibus taxi loading facilities
4. Report on agent-based simulation
5. Identify the agents that play a role in a minibus taxi loading facility
6. Research and model their behaviour and decision-making framework
7. Research and model the environment these agents function in
8. Create a simulation model incorporating agents present at taxi ranks
9. Compare the created simulation model to existing observations at taxi ranks
10. Determine the validity of the simulation model

Chapter 2 is the result of steps one through four. The knowledge developed in chapter 2 was then used to inform chapters 3 and 4. Chapters 3 and 4 describe how steps five to eight were performed. Finally, chapter 5 describes how steps nine and ten were performed.
The result of performing the sequence of actions decided upon, was a deliberative agent model of a taxi rank, referred to as ACTS. In chapter 5, ACTS was shown to provide valid insights into the operation of a minibus-taxi loading facility. ACTS is designed so that changes to the design or operation of a minibus-taxi loading facility can be simulated. It can therefore provide answers to "what if?" questions posed about minibus-taxi loading facilities. The sequence of actions performed, therefore, did lead to the desired goal state.

6.2 Reflection on solution

The object-oriented approach used ensured that the taxi rank system was decomposed into individual components. This makes the system modular. The components can be studied, adjusted and improved in isolation. The components are represented by the classes and algorithms developed in chapter 4. The use of deliberation by agents in taxi rank simulation is novel. The use of GOAP makes it straightforward to model the goals and actions available to the various role-players at a taxi rank. In future, more role-players can be identified, modelled and included in the simulation model. The solution developed is also an example of "digital twinning". Digital twinning refers to the creation of virtual replicas of the real world (I. Venter 2019). The United Kingdom Rail Research and Innovation Network (UKRRIN) were able to use digital twinning to decrease the delays in the 200 km railway system between Birmingham and London from 2300 seconds to 800 seconds. This was done by using digital twinning to quickly create and test powerful "what-if" scenarios. The ACTS system can also be used to create and test "what-if" scenarios.

6.3 Recommendations

The techniques used in this project, can find applications at various levels of abstraction. The first level is to consider the techniques as applied: A simulation model of a taxi rank where the deliberation of agents about the taxi rank environment lead to actions by those agents. The actions of the agents cause changes to the simulation environment which can then be measured. Outcomes at this level include:

- The lengths of all commuter queues at any time-step throughout the simulation
- The longest commuter queue that formed during the simulation

- The distances that commuters generally walk in the simulation
- The most direct paths that commuters can take in the rank as affected by the rank geometry and movement of minibus taxis
- The percentage of passengers that the facility could serve during the simulation
- The average time that commuters must wait before departing the rank
- The longest queue of taxis that occurred during the simulation
- The frequencies and locations of conflicts between commuters and the minibus taxis in the rank
- The effects of changing the geometry of the rank on the above-mentioned variables
- The effects of changing the operational layout of the rank on the above-mentioned variables
- The effects of changing driver behaviour on the above-mentioned variables
- The effects of changing commuter behaviour on the above-mentioned variables
- Identifying the points of conflict between vehicles and pedestrians
- An evaluation of the functional performance of the taxi rank design regarding the above variables. Some of the variables are indicators of economic efficiency. Others are indicators of the experience the rank provides to the commuters and drivers.
- The ability to test and compare ideas before investing in infrastructure.

At a higher level, the techniques used can be considered as the creation of a simulation model that represents the interaction of various agents that want to navigate a transportation environment with that environment and each other. Measurements done during simulation provide insights into the functioning of that environment. Proposed examples include:

- Simulation of a mini-bus taxi facility
- Simulation of a local road network
- Simulation of an airport terminal
- Simulation of a train station
- Simulation of a university campus

Through a lens at a yet higher level, it becomes evident that transportation is not the only field in Civil Engineering that may find benefit from applying the techniques used in this project. At this level, the model is seen as representing any combination of decision-making agents and environments. The goals of the agents do not have to be confined to "reach destination" as in transportation. Some proposed examples include:

- Simulation of a construction site and how different agents may interact to fulfil the common goal of completing the project
- Simulation of the use of a building by agents. This may include normal operation, various emergency scenarios and other situations that insights are needed for.
- Simulation of water usage in a building or neighbourhood.
- Simulation of electricity usage in a building or neighbourhood.

At the highest level of abstraction considered, the techniques used are an attempt at reflecting reality and possible realities. Seeing it at this level reveals opportunities to:

- Allowing human input into the simulation. This would bring true to life decision making into the simulation.
- Deepening immersion and the validity of human input by using techniques such as Virtual Reality, Augmented Reality and haptic feedback systems.
- Connecting live data feeds to the simulation.
- Communicate knowledge and information to an audience (usually the client)

Perhaps a further abstraction would be to consider supposed impossible realities and in doing so, shed light on unimagined possibilities.

References

Ahmed, Kazi Iftekhar (1999). "Modeling Drivers ' Acceleration and Lane Changing Behavior". In: *Transportation* Ph.D, p. 189. URL: https://its.mit.edu/sites/default/files/documents/DRIVIN.PDF%20http://web.mit.edu/its/papers/DRIVIN.PDF.

Barcelo, Jaume (2010). *Fundamentals of Traffic Simulation (International Series in Operations Research & Management Science)*, p. 460. ISBN: 9781441961419.

Cervero, Robert and Aaron Golub (2007). "Informal transport: A global perspective". In: *Transport Policy* 14.6, pp. 445–457. ISSN: 0967070X. DOI: 10.1016/j.tranpol.2007.04.011. URL: https://ac.els-cdn.com/S0967070X07000546/1-s2.0-S0967070X07000546-main.pdf?%7B%5C_%7Dtid=8ce7bc7c-1b68-4e5e-b3b0-63cad7fc126b%7B%5C&%7Dacdnat=1520234010%7B%5C_%7D.

Chen, S et al. (1995). "CAR-FOLLOWING MEASUREMENTS, SIMULATIONS, AND A PROPOSED PROCEDURE FOR EVALUATING SAFETY". In: pp. 529–534. ISBN: 9780080423708. DOI: 10.1016/B978-0-08-042370-8.50030-6.

Cocking, L (2017). "A User's Guide to the Mexico City Public Transport System". In:

Dijkstra, Edsger W. (1959). "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1, pp. 269–271.

Edie, Leslie C. (1961). "Car-Following and Steady-State Theory for Noncongested Traffic". In: *Operations Research* 9.1, pp. 66–76.

Fellendorf, Martin and Peter Vortisch (2001). "Validation of the microscopic traffic flow model VISSIM in different real-world situations". In: *transportation research board 80th annual meeting*.

Fobosi, Siyabulela (2013). "The minibus taxi industry in South Africa: A servant for the urban poor?" In:

Fourie, P.J. (2009). "An initial implementation of a multi-agent transport simulator for South Africa". PhD thesis, p. 69. URL: https://repository.up.ac.za/bitstream/handle/2263/25793/dissertation.pdf;sequence=1%20http://repository.up.ac.za/handle/2263/25793.

Fourie, P.J. (2010). *AGENT-BASED TRANSPORT SIMULATION VERSUS EQUILIBRIUM ASSIGNMENT FOR PRIVATE VEHICLE TRAFFIC IN GAUTENG*. Tech. rep. Pretoria, pp. 978–979. URL: https://repository.up.ac.za/bitstream/handle/2263/14799/Fourie%7B%5C_%7DAgent%7B%5C%%7D282010%7B%5C%%7D29.pdf?sequence=1%7B%5C&%7DisAllowed=y.

Garber, Nicholas and Lester Hoel (2015). *Traffic and Highway Engineering*. 5th. Stanford, CA, USA: Cengage Learning.

Gazis, Denos C., Robert Herman, and Renfrey B. Potts (1959). "Car-Following Theory of Steady-State Traffic Flow". In: *Operations Research* 7.4, pp. 499–505.

Gazis, Denos C., Robert Herman, and Richard W. Rothery (1961). "Nonlinear Follow-the-Leader Models of Traffic Flow". In: *Operations Research* 9.4, pp. 545–567.

Gerlough, Daniel L and Matthew J Huber (1975). *Traffic Flow Theory. TRB Special Report 165*. Washington, D.C.: Transportation Research Board. ISBN: 0309024595. URL: http://onlinepubs.trb.org/Onlinepubs/sr/sr165/165.pdf.

Ghallab, Malik, Dana Nau, and Paolo Traverso (2016). *Automated Planning and Acting*, pp. 1–354. ISBN: 9781139583923. DOI: 10.1017/CBO9781139583923. URL: http://projects.laas.fr/planning/book.pdf.

Günes, Mesut (2012). *Verification and Validation of Simulation Models*. DOI: 10.1016/S0076-5392(08)62295-X. arXiv: arXiv:1011.1669v3. URL: https://pdfs.semanticscholar.org/2a81/6a9a5921443504316c19351aed13bef84674.pdf (visited on 06/19/2019).

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2), pp. 100–107.

Hidas, Peter (1998). "A car-following model for urban traffic simulation". In: *Traffic engineering & control* 39.5.

Ingle, Mark (2009). "A historical overview of problems associated with the formalisation of the South African minibus taxi industry". In:

Kantey and Templar Consulting Engineers (2018). *Update to Stellenbosch Comprehensive Integrated Transport Plan*. Stellenbosch Municipality.

Leutzbach, Wilhelm (1988). *Introduction to the theory of traffic flow*. Vol. 47. Springer.

Lieberman, E and A Rathi (2005). "Traffic simulation". In: *United States Transportation Research Board Revised Monograph on Traffic Flow Theory*, pp. 1–23.

Masuku, Thembekile (2016). "Does The Spirit of Ubuntu Exist in the Minibus Taxi Industry: A Form and Function of the Socio-Economic Lives of Queue Marshals in Bree Taxi Rank, Johannesburg?" In: *Development Studies in the School of Social Sciences, University of the Witwatersrand* March, pp. 1–127.

URL: http://wiredspace.wits.ac.za/jspui/bitstream/10539/21841/1/final%20submission.pdf.

Mathew, Tom (2017). *Microscopic Traffic Simulation 37.1 Overview*. Tech. rep. Bombay. URL: https://www.civil.iitb.ac.in/tvm/nptel/tselnp37.pdf.

May, Adolf D and Hartmut E M Keller (1967). *Non-Integer Car-Following Models*. Tech. rep. Berkeley. URL: http://onlinepubs.trb.org/Onlinepubs/hrr/1967/199/199-002.pdf.

Neumann, Andreas, Daniel Röder, and Johan W Joubert (2015). "Towards a simulation of minibuses in South Africa". In: *Journal of Transport and Land Use* 8.1, p. 137. DOI: 10.5198/jtlu.2015.390. URL: http://dx.doi.org/10.5198/jtlu.2015.390.

Newell, Gordon Frank (2002). "A simplified car-following theory: a lower order model". In: *Transportation Research Part B: Methodological* 36.3, pp. 195–205.

Orkin, Jeff. (2006). "Three states and a plan: the AI of FEAR". In: *Game Developers Conference* 2006. URL: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.

Parker, MT (1996). "THE EFFECT OF HEAVY GOODS VEHICLES AND FOLLOWING BEHAVIOUR ON CAPACITY AT MOTORWAY ROADWORK SITES." In: *Traffic engineering & control*.

Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press. ISBN: 0136042597, 9780136042594.

Sargent, R. G. (2013). "Verification and validation of simulation models". In: *Journal of Simulation* 7.1, pp. 12–24. ISSN: 17477778. DOI: 10.1057/jos.2012.20. arXiv: 10 [978-1-4244-9864-2]. URL: https://pdfs.semanticscholar.org/fd6a/e52a7fc4558acfb7e77d029e52431c11a3eb.pdf.

South African Department of Transport (2007). *Guidelins for the Design of Mini/Midibus - Taxi Facilities*.

Statistics South Africa (2017). *General Household Survey*. STATSSA.

Van Der Merwe, Janet (2011). *AGENT-BASED TRANSPORT DEMAND MODELLING FOR THE SOUTH AFRICAN COMMUTER ENVIRONMENT*. Tech. rep. Pretoria: University of Pretoria. URL: https://repository.up.ac.za/bitstream/handle/2263/23197/dissertation.pdf;sequence=1.

van-Biljon, B and CJ Venter (2013). "The use of Monte Carlo simulation to determine the optimal configuration for minibus taxi loading areas." In: *Citeseer* July, pp. 234–244. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.877.640%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

van-Dalsen, A (2018). "Brief: Minibus Taxis". In:

Venter, Christoffel (2013). "The lurch towards formalisation: Lessons from the implementation of BRT in Johannesburg, South Africa". In: *Research in Transportation Economics* 39.1, pp. 114–120. ISSN: 07398859. DOI: 10.1016/j.retrec.2012.06.003. arXiv: 07398859. URL: https://ac.els-cdn.com/S0739885912000753/1-s2.0-S0739885912000753-main.pdf?%7B%5C_%7Dtid=ec7d4517-45e9-455d-8e41-2a814951516c%7B%5C&%7Dacdnat=1520238777%7B%5C_%7D.

Venter, Irma (2019). *Digital twinning, robotic testing, hydrogen trains and the railways*. URL: https://www.engineeringnews.co.za/article/digital-twinning-robotic-testing-hydrogen-trains-and-the-railways-2019-07-30/rep%7B%5C_%7Did:4136 (visited on 08/09/2019).

Vural, Z (2019). "Getting Around in Nairobi". In:

Wevell, Dennis (2011). "Implementing MATSim Transit Simulation in a South African context". In: October. URL: https://repository.up.ac.za/bitstream/handle/2263/17986/Wevell%7B%5C_%7DImplementing(2011).pdf?sequence=1.

Wiedemann, Rainer (1974). *Simulation des Strassenverkehrsflusses*. Tech. rep. Institut fur Verkehrswesen.

Woolf, S E and J W Joubert (2013). "A people-centred view on paratransit in South Africa". In: *Cities* 35, pp. 284–293. ISSN: 02642751. DOI: 10.1016/j.cities.2013.04.005. URL: http://dx.doi.org/10.1016/j.cities.2013.04.005.

Appendix A: Plan of Bergzicht taxi rank

NOTE:
NEW LANDSCAPING AND TREES NOT SHOWN ON THIS DRAWING REFER TO LANDSCAPING DRAWING

SERVICES	EXISTING	PROPOSED
ELECTRICAL CABLES	E	E
ELECTRICITY DUCTS	E	E
TRAFFIC SIGNAL DUCTS	TS	TS
STREET LIGHTING DUCTS	SL	SL
STREET LIGHT CABLE	SL LP	SL LP
TELKOM UNDERGROUND CABLES	T	T
TELKOM DUCTS	T	T
OVERHEAD TELEPHONE	T	T
TELKOM MANHOLE		
WATER MAINS	W	W
WATERING POINT & STOP VALVE	W	W
FIRE HYDRANT	W	W
IRRIGATION DUCTING	IR	IR
FOUL SEWERS	FS	FS
STORMWATER SEWERS	SW	SW
EXISTING SERVICE TO BE ABANDONED OR REMOVED		
UNKNOWN MANHOLES	EX. MH	

EX. BUILDING
EX. BUILDING
NEW SEWER TO TIE INTO EXISTING MH
EXIST. SW PIPE TO BE REMOVED
EXIST. SWMH TO BE REMOVED
PALISADE FENCE
GATE
WHEATSTONE BRICK PAVING HERRINGBONE PATTERN
PARKING BAYS DEMARCATED WITH GREY COBBLES
UPGRADED PARKING AREA
PLANTER
BRICK PAVING
BIN WITH SURROUND
ELECTRICAL JUNCTION BOX AS PER DETAIL ON DWG C1040-W-C-021
GREASE TRAP AS PER DETAILS
NEW MANHOLE TO BE BUILT ON EXISTING SEWER MAIN
110Ø uPVC CLASS 34 SEWER PIPE
FS GULLY GRADIAN HORIZONTAL MORE TO CENTRE OF MORRAT FLOOR SLAB
2 x 160Ø
DEMOLISH EXISTING LOW WALLS
2G INLET MH
REMOVE BOLLARDS
DEMOLISH EXISTING LOW WALLS
2G INLET MH
PAVEMENT
MERRIMAN AVENUE
BIRD STREET
EXIST. SWMH 5
CL +154.287
IL +154.287

Revisions			
No.	Date	Description	Initial
00	2010/09/30	TENDER DRAWING ONLY	P NEL

for VELA VKE Consulting Engineers (Pty) Ltd.
date

for CLIENT
date

Designed By	P NEL
Checked By	G WOODS
Drawn By	P NEL
Checked By	G WOODS

SABS
VELA VKE
CONSULTING ENGINEERS (PTY) LTD
CESA Member
PO Box 8022
Roggebaai 8012
email capetown@velavke.co.za
website www.velavke.co.za
65 Riebeek St
Cape Town 8001
Tel +27 (021) 417-2900
Fax +27 (021) 417-2999
MM Incorporating Munyuli Malikula Engineers (Pty) Ltd
Bloemfontein • Cape Town • Durban • East London • George • Johannesburg • Kimberley • Mossel Bay • Nelspruit • Polokwane • Port Elizabeth • Pretoria • Rustenburg

STELLENBOSCH MUNICIPALITY

UPGRADING OF BERGZICHT TAXI RANK AND ASSOCIATED AREAS

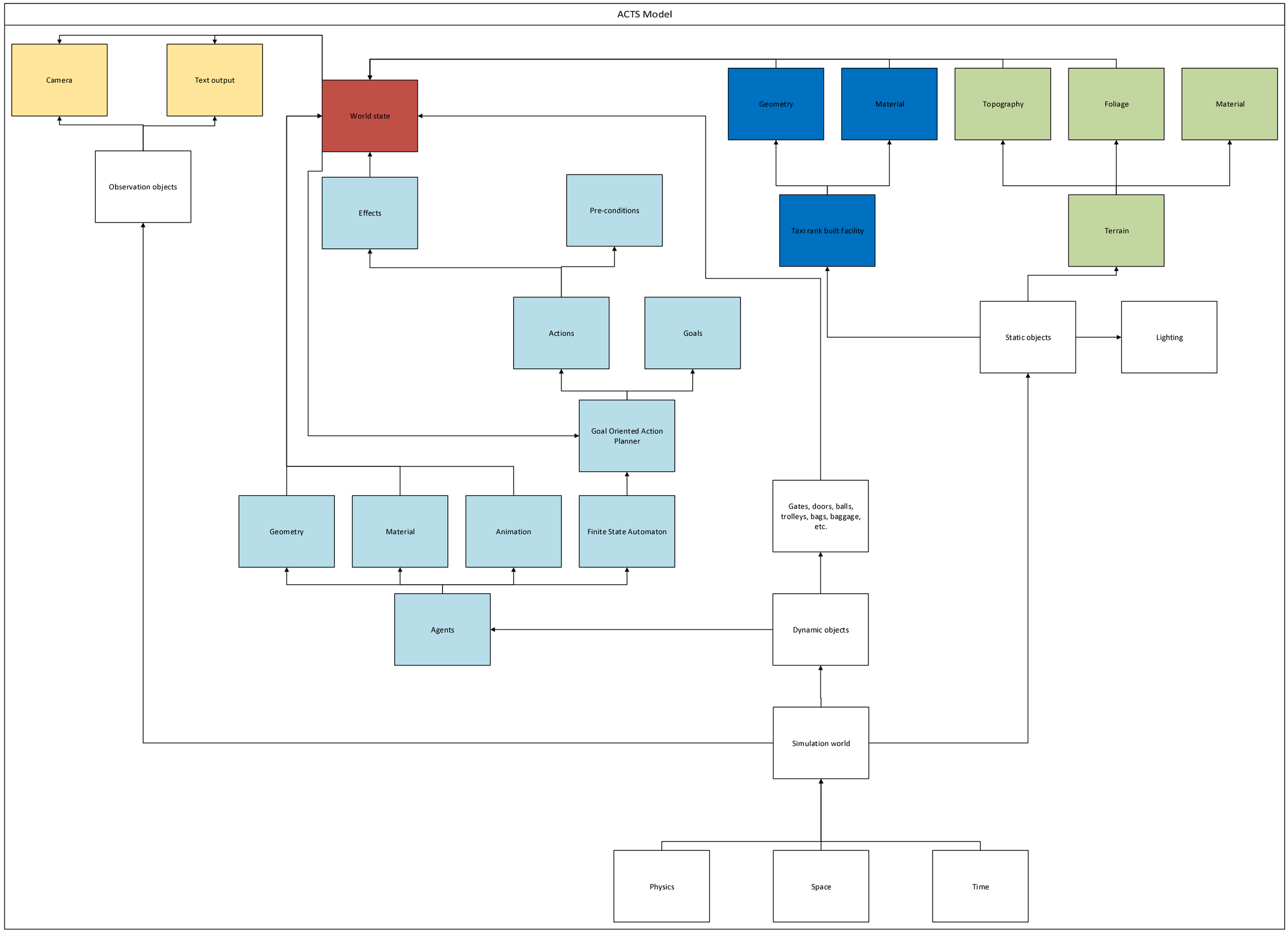
PHASE 3

EXISTING AND PROPOSED SERVICES

A1 Scale 1:250
VELA VKE drawing no.
C1040-W-C-300
Client drawing no.
Revision No. 00

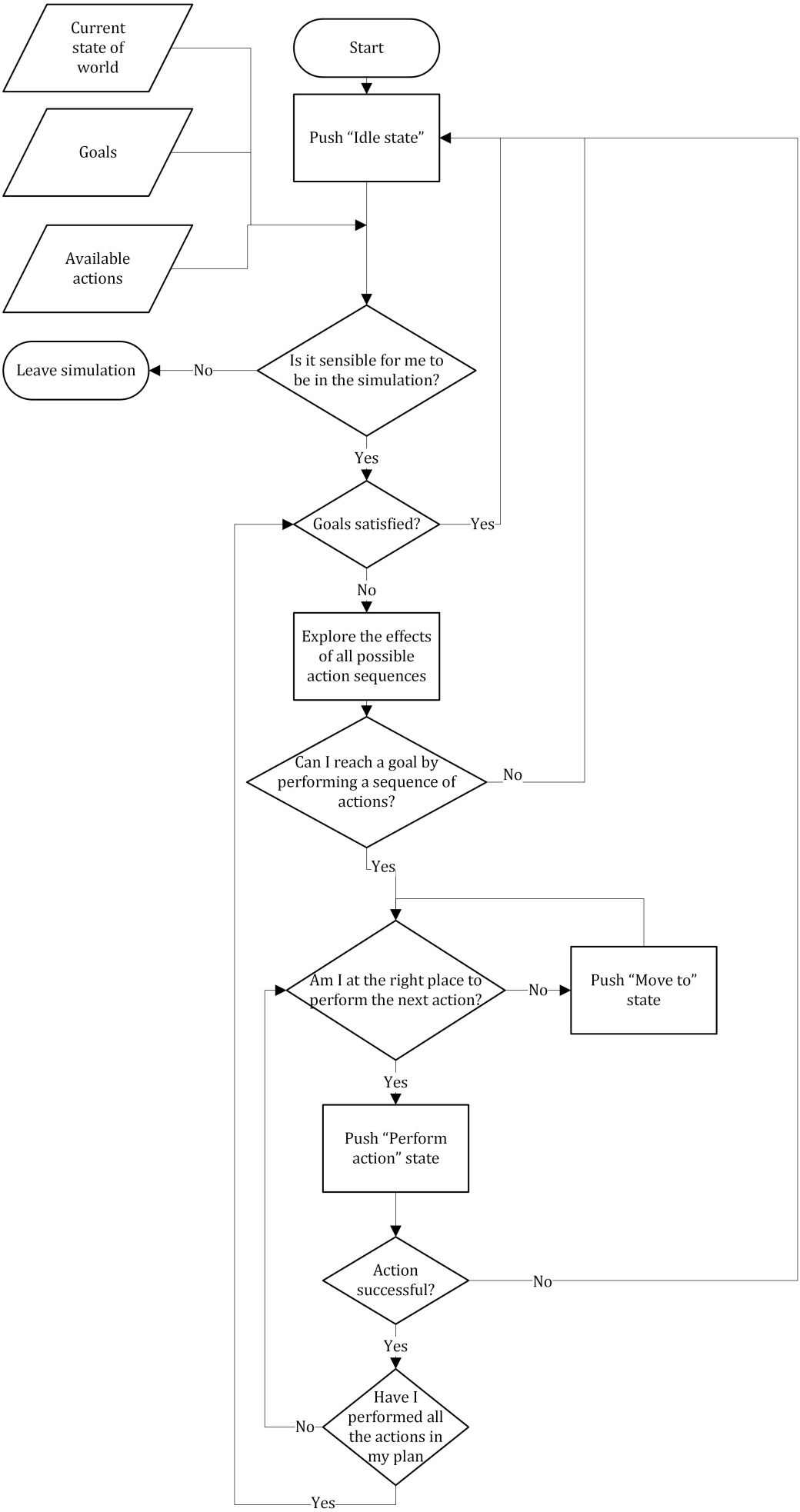
P:\C1040_Bergzicht Taxi Rank\3_Working\3-5_Civil\Drawings\ePASE 3\Phase 3 layout.dwg 2010/10/15 11:52:10 AM

Appendix B: ACTS model



Appendix C: Goap as implemented in ACTS

Agent State Control



Appendix D: Taxi ranks observed

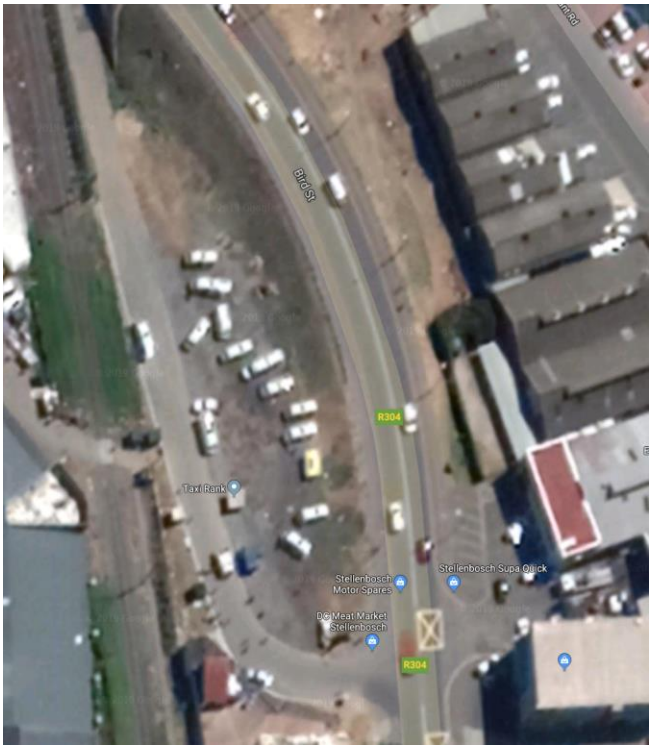
To develop an intuition for the functioning of taxi ranks, the following site visits were performed:

- Observations at the Kayamandi informal taxi rank in Stellenbosch
- Observations at the BT Ngebs taxi rank in Mthatha
- Observations at the Circus Triangle taxi rank in Mthatha
- Observations at the Chatham street rank in Mthatha
- Observations at the Bergzicht taxi rank in Stellenbosch

A summary of the observations made at each of these ranks as well as photographs can be found in the sections below.

Kaymandi informal taxi rank

This taxi rank is located just west of Bird street in Stellenbosch and just south of Kayamandi. As it is an informal rank, there is no built infrastructure. The rank is sandwiched between a rail line and the R304 as can be seen in the figure below:



Photographs taken during the site visit can be found by visiting the link below:

https://photos.app.goo.gl/zwujN7JVfNo8juDcA

The photograph below shows the entirety of the rank as well as highlighting the potential conflict areas between pedestrians and the trainline. Also visible, to left of the taxi rank, is the R304. Pedestrian desire lines in the grass show that pedestrians traverse a steep slope to reach the sidewalk next to the R304.

The figure below shows that the trains cross at the level of the pedestrian thoroughfare that leads from the taxi rank to Khayamandi.

For a more detailed investigation into safety of the rank as well as pedestrian safety along the R304/Bird street, readers are referred to the following document:

https://drive.google.com/open?id=1OZSmYi0gZSvybAxWIXBRkVn8hMzkPYP1

BT Ngebs taxi rank

This taxi rank is next to the BT Ngebs mall in Mthatha as can be seen in photograph below:

Photographs taken during the site visit can found at the link below:

https://photos.app.goo.gl/fViSD1UU5mHew3QU6

The photograph below shows the general layout of the rank as well as the vantage point from which observations were made.

It could be seen that taxis enter the rank from the eastern side and then stop at the central island to wait for passengers. There were many taxis parked that were not loading passengers during the off-peak times.

It could be seen that the rank essentially operates as a shared space for pedestrians and taxis.

Circus Triangle taxi rank

This taxi rank is located just to the west of the Circus Triangle mall in Mthatha. It has the capacity for larger vehicles such as full-sized buses and is used as an origin for longer haul trips.

An aerial view of the rank can be found in the figure below:

Photographs of the rank can be viewed by visiting the link provided below:

https://photos.app.goo.gl/CzvyGSzGbK6fcfZc9

The photograph below shows the mixed usage of the rank including minibuses, midi-buses, full sized buses, pickup trucks and flatbed trucks:

It could also be seen that washing vehicles was a large part of the daily activities at the rank.

Chatham street rank

This taxi rank is located to the west of Chatham street in Mthatha.

Photographs of the site visit can be found at the link below:

https://photos.app.goo.gl/2dTwAK9SwDvM45XL6

It could be seen that rank is too small for the number of vehicles that wish to make use of the facility, with minibus taxis making use of nearby roads and parking facilities to park when they do not find space at the rank:

Hawking was also seen to be a large part of the activities taking place at the rank.

Many minibuses could be seen waiting for peak times to load passengers.

Bergzicht taxi rank

Bergzicht rank is located on the south western corner of the intersection of Merriman avenue and Bird street in Stellenbosch as shown below:

Commuters can reach the following main destinations from Bergzicht taxi rank:

- Kylemore/Pniel
- Cloetesville
- Idas Valley
- Kayamandi
- Jamestown
- Somerset West

- Vlottenburg
- Devon Valley
- Koelenhof
- Eersterivier

As Bergzicht taxi rank was used as the main case study in this research, multiple extensive site visits were performed:

- A visit during a morning peak time: https://photos.app.goo.gl/hMRwCvccT1P3dsie6
- A visit during midday: https://photos.app.goo.gl/8b5pr5faqdThYu8E6
- A visit during an afternoon peak time: https://photos.app.goo.gl/yUyc4BMHG8GkN9Gv8
- A visit to make use of the transportation services to travel to Franschoek and back: https://photos.app.goo.gl/rL3A8Sm4uiHtvoVi7

Appendix E: Safety at Bergzicht rank

To develop an understanding for the safety challenges present at taxi ranks, especially at Bergzicht taxi rank, a special site visit was performed during an afternoon peak period. This site visit was performed on a Monday afternoon peak period, that is from 15:00 to 18:00.

Photographs from the site visit can be found at the link below:

https://photos.app.goo.gl/yUyc4BMHG8GkN9Gv8

What follows is an account of the site visit by the researcher:

"It was interesting. It went, okay… I got there at about three and started off my observations in the south western corner of the rank. Within 10 minutes a youth was trying to sell me watches. After I graciously declined the offer and continued my observations, the youth remained close by and sat done as if waiting for someone or something. Most definitely interesting… The taxis drove mostly at walking pace. If people were walking in front of the taxis, the taxis would give a little hoot and the people would move off to the side out of the way of the taxi. About half an hour later, I went to the north western corner of the taxi rank that serves as the main entrance for the minibuses. There, the taxis were driving at about double walking pace. There were fewer people that walked in that area and the corridor that the taxis used to drive into the rank was generally kept clear of pedestrians. If pedestrians needed to cross the corridor, they did so as quickly as possible. There were two men sitting to my left during my observations there. I could hear them discussing deep philosophical questions. To my right, a teenager listened to music while he lay on his back on the foot-high concrete wall we were all on. At one stage a minibus taxi stopped in front of us in the corridor. The teenager noticed that the taxi's door was inadvertently ajar. He stood up, walked up to the taxi, closed the door and then went back to lay down on the wall and listen to his music. At about 4 pm, I moved to go observe the eastern side of the taxi rank. I watched the operations of the rank and noticed that there were men wearing blue over vests. On the vests, stood the words "Trolley service" and I thought: "What could this mean? What could be this trolley service?" After watching them for another few minutes, I concluded that they were watching for people that brought shopping trolleys from stores like Checkers and Shoprite Usave. They collected these trolleys and lined them up, ready to be taken back to the respective stores from which they had been taken. Also, at the eastern part of the rank, I observed that there were flatbed trucks parked. A forklift would load or unload pallets of produce such as cabbage heads and pumpkins. This operation was happening in the pedestrian thoroughfare that led from the south eastern part of the rank to the north eastern part of the rank. I did not feel safe walking past the fast swinging forklift in that constricted space. At about 5 pm I walked through the centre of the rank and observed commuters queueing patiently on the parallel islands waiting for taxis to pick them up. Once I reached the western side of the rank, I sat down and observed the taxi rank from that side. Sitting there reinforced my conclusion that the rank operated essentially as a shared space between pedestrians and vehicles. As both parties were aware of the potential danger of conflicts between them, they generally kept to safe speeds and positions. When conflicts arose, they were resolved by a quick word or hoot and the parties involved moved out of each other's ways. While sitting at the western side, a man paced across from my right to my left. He then crossed my vision from left to right, each time inching closer. I could see him take a small bottle out of his pocket, out of which he took a quick sharp sniff before hiding the bottle in his pocket again. In his other hand, he held a small container of rolled cigarettes. Soon, he approached me and asked, "Why are you sitting all alone here?" I replied, "Oh you know just watching the day go by." "Aha," he sniggered, "Just watching the view…" I left to go observe from the northern side of the rank

where there were more people. It was about half past five and the rank had really started to get busy. I stood; mesmerised by the queues forming out of the Kayamandi aisle as my model had predicted. Out of the right edge of my vision I saw a man approaching me holding large cardboard signs that were facing away from me. When he was about 3 metres from me, he suddenly threw them to the ground and continued walking past me. I went to look the posters he had so carelessly discarded. They were posters giving advice for remaining safe on a taxi journey:

The posters were concerned with safety outside of the rank on taxi journeys. After a taxi driver approached me and warned me to be careful in taking photos, I put my camera away and discreetly watched the rank operations from the eastern side of the rank until it was 18:00."

The main conclusions drawn from the observations are as follows:

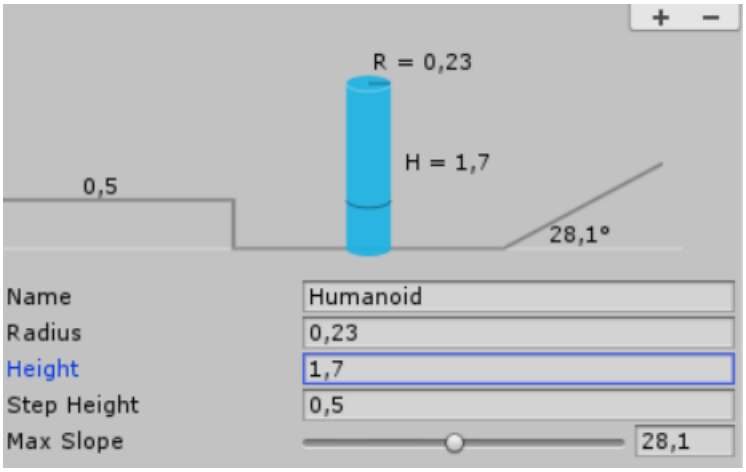
- The taxi rank operates as a shared space for pedestrians and vehicles, ensuring that they act mindful of each other. This means that pedestrian-vehicle conflicts pose a minimal risk within the space of the rank.
- The taxis drive faster near the entrance/exit of the rank. It may be advisable to implement speed control measures to reduce the risks associated with conflicts that arise there.
- There are many different role-players at the rank with different intentions, i.e. goals. Some of these goals may affect the safety of people in the rank. The ACTS model lends itself well to modelling these goals and their effects.
- The researcher experienced higher risks when interacting with fellow pedestrians than with vehicles. Such risks may be fewer when the reason for a visit to the rank is not research.

Appendix F: Universal Access

When using Unity's NavMeshSurface component to determine what surfaces can be navigated by NavMeshAgents, one can set the following parameters:

- Approximate radius of the agent – this will affect how narrow of a gap the agents can pass trough
- Maximum step height – this affects the allowable height of steps that agents can climb
- Maximum slope – this is the maximum slope in degrees that the agent is capable of traversing.

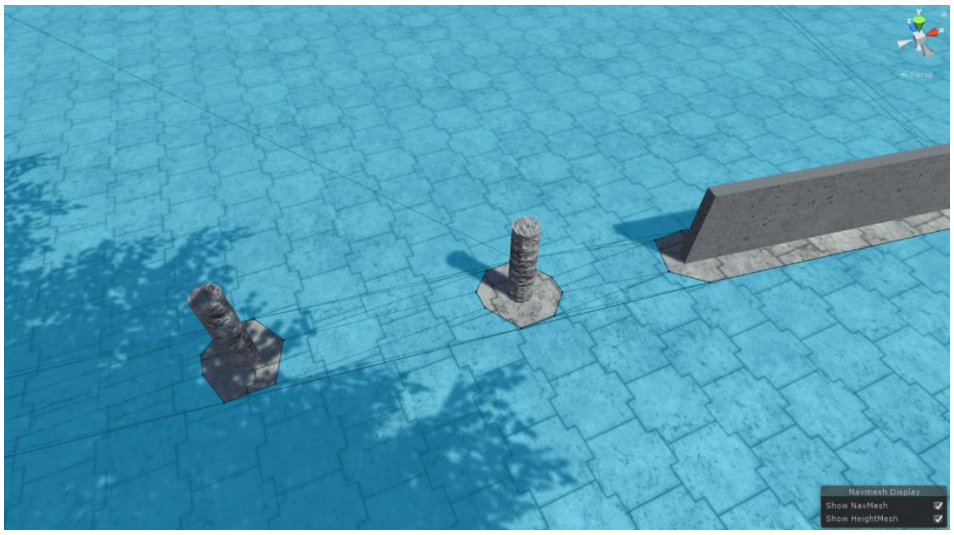
These settings can be seen in the figure below as configured for a humanoid agent as an example:



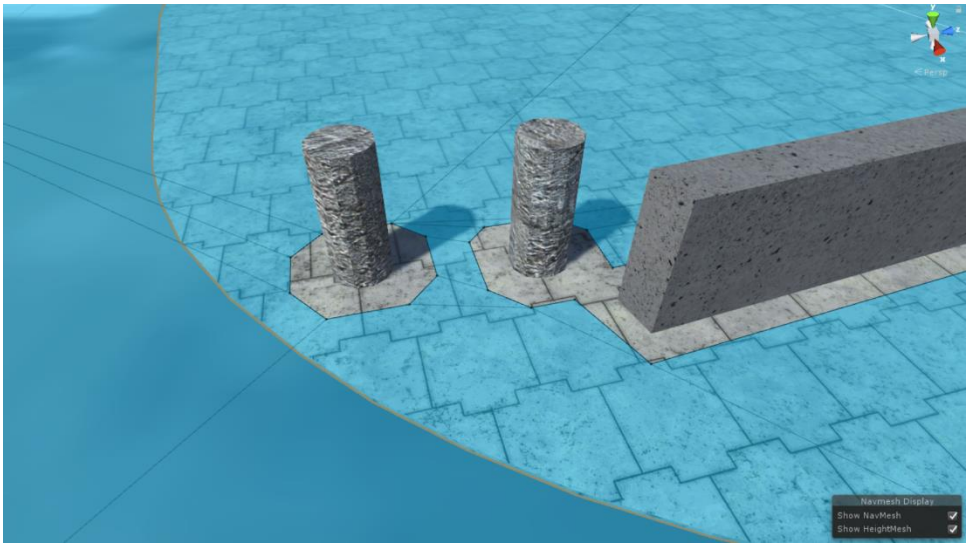
One can set different configurations for various agents such as:

- Wheelchair users
- People with visual impairments
- People who rely on prosthetics for mobility
- Other users of the rank with special mobility needs

Once the configurations have been set, Unity can generate a NavMesh shown in blue below:



The NavMesh shows all the surfaces that were determined to be navigable as per the configuration defined. Gaps in the NavMesh indicate areas that cannot be navigated by the agent under analysis. The figure below shows that there is a bollard placed too close to the wall and that it may make the rank less accessible.



By doing such checks during the design stage, the construction of taxi ranks that are not universally accessible can be avoided.

Appendix G: An experience on a taxi

To develop an understanding for the experiences of commuters when using the services provided by minibus taxis, the researcher set out to embark on a journey by taxi themselves. The journey was undertaken during a Wednesday AM peak period on 6 March 2019.

Photographs of the journey can be seen by following the link below:

https://photos.app.goo.gl/rL3A8Sm4uiHtvoVi7

What follows is an account of the journey by the researcher:

"To get a better picture of what it is like to use minibus taxi services, My German exchange student friend, Simon, and I devised a plan: We would try to travel to Franschoek from Bergzicht rank in Stellenbosch and then back. We arrived at the rank at about seven thirty. After asking the helpful people around the taxi rank about how we could bring our plan to fruition, we ascertained that we would have to start our journey by taking a taxi to Pniel. We waited for about twenty minutes and then got on a taxi that was headed for Kylemore and eventually Pniel. After a circuitous route through Kylemore, we arrived at Pniel at about quarter past eight. There, we got off the taxi and it returned to Stellenbosch while we waited for a taxi that would transport us for the next leg of our journey. After inquiring, we learnt that we would have to get on a taxi travelling to Paarl and then get off it at the T-junction where a left takes you to Paarl and a right takes you to Franschoek. At eight fifty, such a taxi arrived, and we got onto it. By nine, we were at the T-junction and did not have to wait long before we could get onto an old Toyota Hi-Ace travelling from Paarl to Franschoek. The atmosphere on the Hi-Ace was lively with upbeat music playing. Just before we got to Franschoek, the taxi made a left turn up a hill into an informal settlement. We asked the driver where we were going, and he told us that he was looking to pick up passengers in the area known as "The Neighbourhood". As we drove slowly up the hill into the neighbourhood, the driver kept lightly tapping on his hooter to let people know that he was ready to pick up passengers. At the top of the hill as we turned around, a child ran up to the window of the taxi and offered Simon a euro cent. Travelling down the hill was at the same slow pace as up the hill and some more passengers boarded the taxi. The driver then proceeded to Franschoek and stopped next to the Pick 'n Pay. The people from the neighbourhood, as well as Simon and I, then got off the taxi. We explored Franshoek for an hour or so before returning to the Pick 'n Pay where we caught a lift on another taxi. This taxi then proceeded to the neighbourhood and we had a repeat of our previous experience there except that more people were getting off the taxi as apposed getting on the taxi. The taxi then proceeded to travel to Paarl and we got off at the intersection to Stellenbosch. After about 20 minutes of waiting, a taxi heading to Pniel picked us up. In Pniel, we transferred to a taxi heading to Stellenbosch and arrived back in Stellenbosch at quarter to one in the afternoon."

The main conclusions drawn from the observations made by the researcher are as follows:

- Most of commuter's time in the taxi system is spent waiting.
- Knowledge about how to travel somewhere spreads by word of mouth at the taxi ranks.
- Generally, the passengers on taxis are open to questions and conversation.
- For unusual routes, many transfers are necessary, and the irregular schedules of the taxis makes it hard to judge when you will arrive at your destination.
- Taxi fares are paid at the end of the journey, usually to the driver or front passenger.